TutorialsPoint 微软技术教程

wizardforcel

Published with GitBook



目錄

TutorialsPoint 微软技术教程

W3School ASP教程

来源：www.w3school.com.cn 整理：飞龙 日期：2014.9.30

ASP 基础

ASP 简介

ASP 文件可包含文本、 **HTML** 标签和脚本。**ASP** 文件中的脚本可在服务器上执行。

在学习之前，应具备的知识：

在继续学习之前，您需要对以下知识有基本的了解：

- HTML / XHTML
- 脚本语言，比如 JavaScript 或者 VBScript

如果希望学习上面的项目，请在我们的 首页 访问这些教程。

ASP 是什么？

- ASP 指 Active Server Pages （动态服务器页面）
- ASP 是一项微软公司的技术
- ASP 是在 IIS 中运行的程序
- IIS 指 Internet Information Services （Internet 信息服务）
- IIS 是 Windows 2000 及 Windows 2003 的免费组件
- IIS 同时也是 Windows NT 4.0 的可选组件
- 此可选组件可通过因特网下载
- PWS 的体积更小 - 不过拥有 IIS 的完整功能
- PWS 可在 Windows 95/98 的安装 CD 中找到

ASP 兼容性

- 运行 IIS，需要 Windows NT 4.0 或更高的版本。
- 运行 PWS， 需要 Windows 95 或者更高的版本。
- ChiliASP 是一种在非 Windows 操作系统上运行 ASP 的技术
- InstantASP 是另一种在非 Windows 操作系统上运行 ASP 的技术

ASP 文件是什么？

- ASP 文件和 HTML 文件类似
- ASP 文件可包含文本、HTML、XML 和脚本
- ASP 文件中的脚本可在服务器上执行。
- ASP 文件的扩展名是 ".asp"

ASP 和 HTML 有何不同？

- 当浏览器请求某个 HTML 文件时，服务器会返回这个文件
- 而当浏览器请求某个 ASP 文件时，IIS 将这个请求传递至 ASP 引擎。ASP 引擎会逐行地读取这个文件，并执行文件中的脚本。最后，ASP 文件将以纯 HTML 的形式返回到浏览器。

ASP 能为你做什么？

- 动态地编辑、改变或者添加页面的任何内容
- 对由用户从 HTML 表单提交的查询或者数据作出响应
- 访问数据或者数据库，并向浏览器返回结果
- 为不同的用户定制网页，提高这些页面的可用性
- 用 ASP 替代 CGI 和 Perl 的优势在于它的简易性和速度
- 由于 ASP 代码无法从来浏览器端察看，ASP 确保了站点的安全性。
- 优秀的 ASP 编程可将网络负载降至最低

重要事项：由于 ASP 在服务器上运行，浏览器无需支持客户端脚本就可以显示 ASP 文件！

在自己的 PC 上运行 ASP

您可以在自己的 **PC** 上运行 **ASP** 。

把自己的 Windows PC 作为 Web 服务器

如果您安装了 IIS 或者 PWS，就可以把自己的 PC 配置为一台 web 服务器。

IIS 或 PWS 可以把您的计算机转变为 web 服务器。

微软的 IIS 和 PWS 是免费的 web 服务器组件。

IIS - Internet Information Server

IIS 是一个基于因特网的服务的集合，由微软开发，在 Windows 平台上使用。

Windows 2000、XP、Vista 以及 Windows 7 均提供 IIS。Windows NT 也可用 IIS。

IIS 很容易安装，是开发和测试 web 应用程序的理想工具。

PWS - Personal Web Server

PWS 用于更老的 Windows 系统，比如 Windows 95、98 以及 NT。

PWS 很容易安装，可用于开发和测试包含 ASP 的 web 应用程序。

我们不推荐使用 PWS，除非是用于培训。它已经过时，并存在安全问题。

Windows Web 服务器版本

Windows 版本	是否提供或支持
Windows 7（所有版本）	提供 IIS 7.5
Windows Server 2008	提供 IIS 7
Windows Vista Business, Enterprise 以及 Ultimate	提供 IIS 7
Windows Vista Home Premium	提供 IIS 7
Windows Vista Home Edition	不支持 PWS 或 IIS
Windows Server 2003	提供 IIS 6
Windows XP Professional	提供 IIS 5.1
Windows XP Home Edition	不支持 IIS 或 PWS
Windows 2000 Professional	提供 IIS 5.0
Windows NT Professional	提供 IIS 3，并支持 IIS 4
Windows NT Workstation	支持 PWS 和 IIS 3
Windows ME	不支持 PWS 或 IIS
Windows 98	提供 PWS
Windows 95	支持 PWS

如何在 **Windows 7** 及 **Windows Vista** 上安装 **IIS**

请根据以下四个步骤来安装 IIS：

1. 从开始菜单打开控制面板
2. 双击“程序和功能”
3. 点击“打开或关闭 Windows 功能”
4. 选择 “Internet 信息服务”的复选框，然后点击确定

在您安装完成 IIS 之后，请确保安装所有补丁包（运行 Windows 更新）。

如何在 **Windows Server 2003** 上安装 **IIS** 并运行 **ASP**

1. 当你启动 Windows Server 2003 后，会看到服务器管理向导
2. 如果向导没有显示，可以打开管理工具，然后选择“配置您的服务器向导”
3. 出现提示后，点击下一步
4. 随后会出现一个“预备步骤”的提示，点击下一步，随后系统自动搜索已在本机安装了的系统服务组件
5. 在随后出现的服务器角色窗口中选择应用程序服务器，然后点击下一步
6. 选中启用*ASP.NET*

7. 随后向导会提示这个组件的大概安装过程，请点击下一步
8. 现在，向导会请求 *Server 2003 CD*。请插入CD后继续运行向导。
9. 最后，向导会提示“此服务器目前是一台应用程序服务器”。
10. 点击完成后，你会发现“应用程序服务器”已在管理你的服务器角色窗口中出现
11. 点击“管理此应用程序服务器”会打在应用程序服务器管理控制台*(MMC)*
12. 展开*Internet* 信息服务 *(IIS)*管理器，然后展开你的服务器，然后是站点文件夹。
13. 你会看到默认的网站，并且它的状态应该是运行中。
14. 在Internet 信息服务 (IIS)管理器中点击“*Web*服务扩展”，你会看到 Active Server Pages 是被禁止的。
15. 选中*Active Server Pages*，然后点击允许按钮，这样 ASP 就被激活了！

如何在 **Windows 2000** 上安装 **IIS** 并运行**ASP**

1. 开始按钮 - 设置 - 控制面板
2. 在控制面板中选择添加/删除程序
3. 在添加/删除程序中选择添加/删除*Windows*组件
4. 在向导窗口中选中 *Internet* 信息服务，然后点击确定
5. *Inetpub* 文件夹会在硬盘中被创建
6. 打开 Inetpub 文件夹，找到名为 *wwwroot* 的文件夹
7. 在 wwwroot 下创建一个新文件夹，比如 "MyWeb"
8. 使用文本编辑器编写几行 ASP 代码，将这个文件取名为 "test1.asp" 保存在 "MyWeb" 文件夹。
9. 确保你的服务器正在运行 - 安装程序会在系统托盘创建一个IIS的图标。点击这个图标，然后在出现的新窗口中按开始按钮。
10. 打开浏览器，在地址栏键入 "http://localhost/MyWeb/test1.asp"，就可以看到你的第一个 ASP 页面了。

如何在 **Windows XP Professional** 上安装 **IIS** 并运行 **ASP**

1. 在 CD-Rom 驱动器中插入 Windows XP Professional CD-Rom
2. 开始菜单，设置，控制面板
3. 在控制面板选择添加/删除程序
4. 在添加/删除程序中选择添加/删除 *Windows* 组件
5. 在向导窗口中选中 *Internet Information Services*，然后点击确定
6. *Inetpub* 文件夹会在硬盘中创建
7. 打开 Inetpub 文件夹，找到名为 *wwwroot* 的文件夹
8. 在 wwwroot下创建一个新文件夹，比如 "MyWeb"
9. 使用文本编辑器编写几行 ASP 代码，将这个文件取名为 "test1.asp" 保存在 "MyWeb" 文

件夹。
10. 确保你的服务器正在运行，使用下面的方法确认它的运行状态：进入控制面板，然后是管理工具，然后双击“*IIS* 管理器”图标。
11. 打开浏览器，在地址栏键入 "http://localhost/MyWeb/test1.asp"，就可以看到你的第一个 ASP 页面了。

提示：Windows XP Home Edition 无法运行 ASP。

如何在老版本的 Windows 操作系统中运行 ASP

如何在 Windows 95 上安装 PWS 并运行 ASP

Windows 95 不包含 PWS！！

要想在 Windows 95 上运行 ASP，就必须从微软的站点下载 “Windows NT 4.0 Option Pack”。

下载"Windows NT 4.0 Option Pack"

如何在 Windows NT 上安装 PWS 并运行 ASP

Windows NT 同样不包含 PWS！！

要想在 Windows NT 上运行 ASP，就必须从微软的站点下载 “Windows NT 4.0 Option Pack”。

下载"Windows NT 4.0 Option Pack"

如何在 Windows 98 上安装 PWS 并运行 ASP

1. 打开 Windows 98 CD 上的 Add-ons 文件夹，找到PWS文件夹并运行其中的 setup.exe 文件。
2. 安装程序会在硬盘创建一个 Inetpub 文件夹。打开这个文件夹，找到 wwwroot 文件夹。
3. 然后在 wwwroot 文件夹下面创建一个新的文件夹，比如 "MyWeb"。
4. 使用文本编辑器编写几行 ASP 代码，将这个文件取名为 "test1.asp" 保存在 "MyWeb" 文件夹。
5. 确保你的服务器正在运行 - 安装程序会在系统托盘创建一个PWS的图标。点击这个图标，然后在出现的新窗口中按开始按钮。
6. 打开浏览器，在地址栏键入 "http://localhost/MyWeb/test1.asp"，就可以看到你的第一个 ASP 页面了。

如何在 Windows ME 上安装 PWS 并运行 ASP

Windows ME 同样不包含 PWS！！

阅读微软站点的信息

来自 Bill James 的安装方法。

ASP 语法

在浏览器中通过查看源代码的方式是无法看到 ASP 源代码的，你只能看到由 ASP 文件输出的结果，而那些只是纯粹的 HTML 而已。这是因为，在结果被送回浏览器前，脚本已经在服务器上执行了。

在我们的 ASP 教程中，每个例子都提供隐藏的 ASP 代码。这样会使您更容易理解它们的工作原理。

实例

用 ASP 写文本

如果使用 ASP 生成文本。

```
<html>
<body>

<%
response.write("Hello World!")
%>

</body>
</html>
```

向文本添加 HTML

如果同时生成 HTML 标签和纯文本。

```
<html>
<body>
<%
response.write("<h2>您可以使用 HTML 标签来格式化文本</h2>")
%>

<%
response.write("<p style='color:#0000ff'>这段文本的样式是通过 style 属性添加的。</p>")
%>
</body>
</html>
```

基本的 ASP 语法规则

通常情况下，ASP 文件包含 HTML 标签，类似 HTML 文件。不过，ASP 文件也能够包含服务器端脚本，这些脚本被分隔符 <% 和 %> 包围起来。

服务器脚本在服务器上执行，可包含合法的表达式、语句、或者运算符。

向浏览器写输出

response.write 命令用来向浏览器写输出。下面的例子向浏览器传送了一段文本："Hello World"。

```
<html>
<body>
<%
response.write("Hello World!")
%>
</body>
</html>
```

还有一种 response.write 命令的简写方法。下面的例子和上面的例子是等效的：

```
<html>
<body>
<%="Hello World!"%>
</body>
</html>
```

在 ASP 中使用 VBScript

你可以在 ASP 中使用若干种脚本语言。不过，默认的脚本语言是 VBScript：

```
<html>
<body>
<%
response.write("Hello World!")
%>
</body>
</html>
```

上面的例子向文档的 body 部分写入了文本 "Hello World!"。

提示：如果您需要了解更多有关 VBScript 的知识，请学习我们的 VBScript 教程。

在 ASP 中使用 JavaScript

如果需要使用 JavaScript 作为某个特定页面的默认脚本语言，就必须在页面的顶端插入一行语言设定：

```
<%@ language="javascript"%>
<html>
<body>
<%
Response.Write("Hello World!")
%>
</body>
</html>
```

注意：与 VBScript 不同 - JavaScript 对大小写敏感。所以你需要根据 JavaScript 的需要使用不同的大小写字母编写 ASP 代码。

提示：如果您需要了解更多有关 JavaScript 的知识，请学习我们的 JavaScript 教程。

其他的脚本语言

ASP 与 VBScript 和 JScript 的配合是原生性的（JScript 是微软的 JavaScript 实现）。如果你需要使用其他语言编写脚本，比如 PERL、REXX 或者 Python，那就必须安装相应的脚本引擎。

重要事项：因为脚本在服务器端执行，所以显示 ASP 文件的浏览器根本无需支持脚本。

ASP 变量

变量用于存储信息。

假如在子程序之外声明变量，那么这个变量可被 **ASP** 文件中的任何脚本改变。假如在子程序中声明变量，那么当子程序每次执行时，它才会被创建和撤销。

实例：

声明变量

变量用于存储信息。本例演示如何声明变量，为变量赋值，并在程序中使用这个变量

```
<html>
<body>

<%
dim name
name="Donald Duck"
response.write("My name is: " & name)
%>

</body>
</html>
```

声明数组

数组用于存储一系列相关的数据项目。本例演示如何声明一个存储名字的数组。

```
<html>
<body>

<%
Dim fname(5),i
fname(0) = "George"
fname(1) = "John"
fname(2) = "Thomas"
fname(3) = "James"
fname(4) = "Adrew"
fname(5) = "Martin"

For i = 0 to 5
      response.write(fname(i) & "<br />")
Next
%>

</body>
</html>
```

循环生成 HTML 标题

如何循环生成 6 个不同的 HTML 标题。

```
<html>
<body>

<%
dim i
for i=1 to 6
   response.write("<h" & i & ">Header " & i & "</h" & i & ">")
next
%>

</body>
</html>
```

使用 Vbscript 制作基于时间的问候语

本例将根据服务器时间向用户显示不同的消息。

```
<html>
<body>
<%
dim h
h=hour(now())

response.write("<p>" & now())
response.write(" (Beijing Time) </p>")
If h<12 then
   response.write("Good Morning!")
else
   response.write("Good day!")
end if
%>
</body>
</html>
```

使用 JavaScript 制作基于时间的问候语

本例同上，只是语法不同而已。

```
<%@ language="javascript" %>
<html>
<body>
<%
var d=new Date()
var h=d.getHours()

Response.Write("<p>")
Response.Write(d + " (Beijing Time)")
Response.Write("</p>")
if (h<12)
   {
   Response.Write("Good Morning!")
   }
else
   {
   Response.Write("Good day!")
   }
%>
</body>
</html>
```

变量的生存期

在子程序外声明的变量可被 ASP 文件中的任何脚本访问和修改。

在子程序中声明的变量只有当子程序每次执行时才会被创建和撤销。子程序外的脚本无法访问和修改该变量。

如需声明供多个 ASP 文件使用的变量，请将变量声明为 session 变量或者 application 变量。

Session 变量

Session 变量用于存储单一用户的信息，并且对一个应用程序中的所有页面均有效。存储于 session 中的典型数据是姓名、id 或参数。

Application 变量

Application 变量同样对一个应用程序中的所有页面均有效。Application 变量用于存储一个特定的应用程序中所有用户的信息。

ASP 子程序

在 **ASP** 中，你可通过 **VBScript** 和其他方式调用子程序。

实例：

调用使用 VBScript 的子程序

如何从 ASP 调用以 VBScript 编写的子程序。

```
<html>

<head>
<%
sub vbproc(num1,num2)
response.write(num1*num2)
end sub
%>
</head>

<body>
<p>您可以像这样调用一个程序：</p>
<p>结果：<%call vbproc(3,4)%></p>

<p>或者，像这样：</p>
<p>结果：<%vbproc 3,4%></p>
</body>

</html>
```

调用使用 JavaScript 的子程序

如何从 ASP 调用以 JavaScript 编写的子程序。

```
<%@ language="javascript" %>
<html>
<head>
<%
function jsproc(num1,num2)
{
Response.Write(num1*num2)
}
%>
</head>

<body>
<p>
结果：<%jsproc(3,4)%>
</p>
</body>

</html>
```

调用使用 VBScript 和 JavaScript 的子程序

如何在一个 ASP 文件中调用以 VBScript 和 JavaScript 编写的子程序。

```
<html>
<head>
<%
sub vbproc(num1,num2)
Response.Write(num1*num2)
end sub
%>
<script  language="javascript" runat="server">
function jsproc(num1,num2)
{
Response.Write(num1*num2)
}
</script>
</head>

<body>
<p>结果：<%call vbproc(3,4)%></p>
<p>结果：<%call jsproc(3,4)%></p>
</body>

</html>
```

子程序

ASP 源代码可包含子程序和函数：

```
<html>
<head>
<%
sub vbproc(num1,num2)
response.write(num1*num2)
end sub
%>
</head>

<body>
<p>Result: <%call vbproc(3,4)%></p>
</body>

</html>
```

将 <%@ language="*language*" %> 这一行写到 <html> 标签的上面，就可以使用另外一种脚本语言来编写子程序或者函数：

```
<%@ language="javascript" %>
<html>
<head>
<%
function jsproc(num1,num2)
{
Response.Write(num1*num2)
}
%>
</head>

<body>
<p>Result: <%jsproc(3,4)%></p>
</body>

</html>
```

VBScript 与 JavaScript 之间的差异

当从一个用 VBScript 编写的 ASP 文件中调用 VBScript 或者 JavaScript 子程序时，可以使用关键词 "call"，后面跟着子程序名称。假如子程序需要参数，当使用关键词 "call" 时必须使用括号包围参数。假如省略 "call"，参数则不必由括号包围。假如子程序没有参数，那么括号则是可选项。

当从一个用 JavaScript 编写的 ASP 文件中调用 VBScript 或者 JavaScript 子程序时，必须在子程序名后使用括号。

ASP 表单和用户输入

Request.QueryString 和 **Request.Form** 命令可用于从表单取回信息，比如用户的输入。

实例：

使用 method="get" 的表单

如何使用 Request.QueryString 命令与用户进行交互。

```
<html>
<body>
<form action="/example/aspe/demo_aspe_reqquery.asp" method="get">
您的姓名：<input type="text" name="fname" size="20" />
<input type="submit" value="提交" />
</form>
<%
dim fname
fname=Request.QueryString("fname")
If fname<>"" Then
      Response.Write("你好！" & fname & "！<br />")
      Response.Write("今天过得怎么样？")
End If
%>
</body>
</html>
```

使用 method="post" 的表单

如何使用 Request.Form 命令与用户进行交互。

```
<html>
<body>
<form action="/example/aspe/demo_aspe_simpleform.asp" method="post">
您的姓名：<input type="text" name="fname" size="20" />
<input type="submit" value="提交" />
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>"" Then
      Response.Write("您好！" & fname & "！<br />")
      Response.Write("今天过得怎么样？")
End If
%>
</body>
</html>
```

使用单选按钮的表单

如何使用 Request.Form 通过单选按钮与用户进行交互。

```
<html>
<%
dim cars
cars=Request.Form("cars")
%>
<body>
<form action="/example/aspe/demo_aspe_radiob.asp" method="post">
<p>请选择您喜欢的汽车：</p>

<input type="radio" name="cars"
<%if cars="Volvo" then Response.Write("checked")%>
value="Volvo">Volvo</input>
<br />
<input type="radio" name="cars"
<%if cars="Saab" then Response.Write("checked")%>
value="Saab">Saab</input>
<br />
<input type="radio" name="cars"
<%if cars="BMW" then Response.Write("checked")%>
value="BMW">BMW</input>
<br /><br />
<input type="submit" value="提交" />
</form>
<%
if cars<>"" then
   Response.Write("<p>您喜欢的汽车是" & cars & "</p>")
end if
%>
</body>
</html>
```

用户输入

Request 对象可用于从表单取回用户信息。

HTML 表单实例

```
<form method="get" action="simpleform.asp">
<p>First Name: <input type="text" name="fname" /></p>
<p>Last Name: <input type="text" name="lname" /></p>
<input type="submit" value="Submit" />
</form>
```

用户输入的信息可通过两种方式取回：Request.QueryString 或 Request.Form。

Request.QueryString

Request.QueryString 命令用于搜集使用 method="get" 的表单中的值。使用 GET 方法从表单传送的信息对所有的用户都是可见的（出现在浏览器的地址栏），并且对所发送信息的量也有限制。

HTML 表单实例

```
<form method="get" action="simpleform.asp">
<p>First Name: <input type="text" name="fname" /></p>
<p>Last Name: <input type="text" name="lname" /></p>
<input type="submit" value="Submit" />
</form>
```

如果用户在上面的表单实例中输入 "Bill" 和 "Gates"，发送至服务器的 URL 会类似这样：

```
http://www.w3school.com.cn/simpleform.asp?fname=Bill&lname=Gates
```

假设 ASP 文件 "simpleform.asp" 包含下面的代码：

```
<body>
Welcome
<%
response.write(request.querystring("fname"))
response.write(" " & request.querystring("lname"))
%>
</body>
```

浏览器将显示如下：

```
Welcome Bill Gates
```

Request.Form

Request.Form 命令用于搜集使用 "post" 方法的表单中的值。使用 POST 方法从表单传送的信息对用户是不可见的，并且对所发送信息的量也没有限制。

HTML 表单实例

```
<form method="post" action="simpleform.asp">
<p>First Name: <input type="text" name="fname" /></p>
<p>Last Name: <input type="text" name="lname" /></p>
<input type="submit" value="Submit" />
</form>
```

如果用户在上面的表单实例中输入 "Bill" 和 "Gates"，发送至服务器的 URL 会类似这样：

```
http://www.w3school.com.cn/simpleform.asp
```

假设 ASP 文件 "simpleform.asp" 包含下面的代码：

```
<body>
Welcome
<%
response.write(request.form("fname"))
response.write(" " & request.form("lname"))
%>
</body>
```

浏览器将显示如下：

```
Welcome Bill Gates
```

表单验证

只要有可能，就应该对用户输入的数据进行验证（通过客户端的脚本）。浏览器端的验证速度更快，并可以减少服务器的负载。

如果用户数据会输入到数据库中，那么你应该考虑使用服务器端的验证。有一种在服务器端验证表单的好的方式，就是将（验证过的）表单传回表单页面，而不是转至不同的页面。用户随后就可以在同一个页面中得到错误的信息。这样做的话，用户就更容易发现错误了。

ASP Cookie

cookie 常用来对用户进行识别。

实例

Welcome cookie

如何创建欢迎 cookie。

```
<%
dim numvisits
response.cookies("NumVisits").Expires=date+365
numvisits=request.cookies("NumVisits")

if numvisits="" then
   response.cookies("NumVisits")=1
   response.write("欢迎！这是您第一次访问本页面。")
else
   response.cookies("NumVisits")=numvisits+1
   response.write("之前，您已经访问过本页面 ")
   response.write(numvisits & " 次。")
end if
%>
<html>
<body>
</body>
</html>
```

什么是 Cookie?

cookie 常用来对用户进行识别。cookie 是一种服务器留在用户电脑中的小文件。每当同一台电脑通过浏览器请求页面时，这台电脑也会发送 cookie。通过 ASP，您能够创建并取回 cookie 的值。

如何创建 cookie?

"Response.Cookies" 命令用于创建 cookie。

注意：Response.Cookies 命令必须位于 <html> 标签之前。

在下面的例子中，我们会创建一个名为 "firstname" 的 cookie，并向其赋值 "Alex"：

```
<%
Response.Cookies("firstname")="Alex"
%>
```

向 cookie 分配属性也是可以的，比如设置 cookie 的失效时间：

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires=#May 10,2020#
%>
```

如何取回 cookie 的值？

"Request.Cookies" 命令用于取回 cookie 的值。

在下面的例子中，我们取回了名为 "firstname" 的 cookie 的值，并把值显示到了页面上：

```
<%
fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)
%>
```

输出：

```
Firstname=Alex
```

带有键的 cookie

如果一个 cookie 包含多个值的一个集合，我们就可以说 cookie 拥有键（Keys）。

在下面的例子中，我们会创建一个名为 "user" 的 cookie 集。"user" cookie 拥有包含用户信息的键：

```
<%
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Adams"
Response.Cookies("user")("country")="UK"
Response.Cookies("user")("age")="25"
%>
```

读取所有的 cookie

请阅读下面的代码：

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Adams"
Response.Cookies("user")("country")="UK"
Response.Cookies("user")("age")="25"
%>
```

假设您的服务器将所有的这些 cookie 传给了某个用户。

现在，我们需要读取这些 cookie。下面的例子向您展示如何做到这一点（请注意，下面的代码会使用 HasKeys 检查 cookie 是否拥有键）：

```
<html>
<body>

<%
dim x,y

 for each x in Request.Cookies
  response.write("<p>")
  if Request.Cookies(x).HasKeys then
    for each y in Request.Cookies(x)
      response.write(x & ":" & y & "=" & Request.Cookies(x)(y))
      response.write("<br />")
    next
  else
    Response.Write(x & "=" & Request.Cookies(x) & "<br />")
  end if
  response.write "</p>"
next
%>

</body>
</html>
```

输出：

```
firstname=Alex

user:firstname=John
user:lastname=Adams
user:country=UK
user:age=25
```

如何应对不支持 cookie 的浏览器？

如果您的应用程序需要和不支持 cookie 的浏览器打交道，那么您不得不使用其他的办法在您的应用程序中的页面之间传递信息。这里有两种办法：

1. 向 URL 添加参数

您可以向 URL 添加参数：

```
<a href="welcome.asp?fname=John&lname=Adams">
Go to Welcome Page
</a>
```

然后在类似于下面这个 "welcome.asp" 文件中取回这些值：

```
<%
fname=Request.querystring("fname")
lname=Request.querystring("lname")
response.write("<p>Hello " & fname & " " & lname & "!</p>")
response.write("<p>Welcome to my Web site!</p>")
%>
```

2. 使用表单

您还可以使用表单。当用户点击提交按钮时，表单会把用户输入的数据提交给 "welcome.asp" ：

```
<form method="post" action="welcome.asp">
First Name:  <input type="text" name="fname" value="">
Last Name: <input type="text" name="lname" value="">
<input type="submit" value="Submit">
</form>
```

然后在 "welcome.asp" 文件中取回这些值，就像这样：

```
<%
fname=Request.form("fname")
lname=Request.form("lname")
response.write("<p>Hello " & fname & " " & lname & "!</p>")
response.write("<p>Welcome to my Web site!</p>")
%>
```

ASP Session 对象

Session 对象用于存储用户的信息。存储于 **session** 对象中的变量持有单一用户的信息，并且对于一个应用程序中的所有页面都是可用的。

Session 对象

当您操作某个应用程序时，您打开它，做些改变，然后将它关闭。这很像一次对话（Session）。计算机知道您是谁。它清楚您在何时打开和关闭应用程序。但是在因特网上有一个问题：由于 HTTP 地址无法存留状态，web 服务器并不知道您是谁以及您做了什么。

ASP 通过为每位用户创建一个唯一的 cookie 的方式解决了这个问题。cookie 被传送至客户端，它含有可识别用户的信息。这种接口被称作 Session 对象。

Session 对象用于存储关于用户的信息，或者为一个用户的 session 更改设置。存储于 session 对象中的变量存有单一用户的信息，并且对于应用程序中的所有页面都是可用的。存储于 session 对象中的信息通常是 name、id 以及参数。服务器会为每个新的用户创建一个新的 Session，并在 session 到期时撤销掉这个 Session 对象。

Session 何时开始？

Session 开始于：

- 当某个新用户请求了一个 ASP 文件，并且 Global.asa 文件引用了 Session_OnStart 子程序时；
- 当某个值存储在 Session 变量中时；
- 当某个用户请求了一个 ASP 文件，并且 Global.asa 使用 <object> 标签通过 session 的 scope 来例示某个对象时；

Session 何时结束?

假如用户没有在规定的时间内在应用程序中请求或者刷新页面，session 就会结束。默认值为 20 分钟。

如果您希望将超时的时间间隔设置得更长或更短，可以设置 *Timeout* 属性。

下面的例子设置了 5 分钟的超时时间间隔：

```
<%
Session.Timeout=5
%>
```

要立即结束 session，可使用 *Abandon* 方法：

```
<%
Session.Abandon
%>
```

注意：使用 session 时主要的问题是它们该在何时结束。我们不会知道用户最近的请求是否是最后的请求。因此我们不清楚该让 session“ 存活”多久。为某个空闲的 session 等待太久会耗尽服务器的资源。然而假如 session 被过早地删除，那么用户就不得不一遍又一遍地重新开始，这是因为服务器已经删除了所有的信息。寻找合适的超时间隔时间是很困难的。

提示：如果您正在使用 session 变量，请不要在其中存储大量的数据。

存储和取回 session 变量

Session 对象最大的优点是可在其中存储变量，以供后续的网页读取，其应用范围是很广的。

下面的例子把 "Donald Duck" 赋值给名为 username 的 session 变量，并把 "50" 赋值给名为 age 的 session 变量：

```
<%
Session("username")="Donald Duck"
Session("age")=50
%>
```

一旦值被存入 session 变量，它就能被 ASP 应用程序中的任何页面使用：

```
Welcome <%Response.Write(Session("username"))%>
```

上面这行程序返回的结果是: "Welcome Donald Duck"。

也可以在 session 对象中保存用户参数，然后通过访问这些参数来决定向用户返回什么页面。

下面的例子规定，假如用户使用低显示器分辨率，则返回纯文本版本的页面：

```
<%If Session("screenres")="low" Then%>
  This is the text version of the page
<%Else%>
  This is the multimedia version of the page
<%End If%>
```

移除 session 变量

contents 集合包含所有的 session 变量。

可通过 remove 方法来移除 session 变量。

在下面的例子中，假如 session 变量 "age" 的值小于 18，则移除 session 变量 "sale"：

```
<%
If Session.Contents("age")<18 then
  Session.Contents.Remove("sale")
End If
%>
```

如需移除 session 中的所有变量，请使用 RemoveAll 方法：

```
<%
Session.Contents.RemoveAll()
%>
```

遍历 contents 集合

contents 集合包含所有的 session 变量。可通过遍历 contents 集合，来查看其中存储的变量：

```
<%
Session("username")="Donald Duck"
Session("age")=50

dim i
For Each i in Session.Contents
  Response.Write(i & "<br />")
Next
%>
```

结果：

```
username
age
```

如果需要了解 contents 集合中的项目数量，可使用 count 属性：

```
<%
dim i
dim j
j=Session.Contents.Count
Response.Write("Session variables: " & j)
For i=1 to j
  Response.Write(Session.Contents(i) & "<br />")
Next
%>
```

结果：

```
Session variables: 2
Donald Duck
50
```

遍历 StaticObjects 集合

可通过循环 StaticObjects 集合，来查看存储在 session 对象中所有对象的值：

```
<%
dim i
For Each i in Session.StaticObjects
  Response.Write(i & "<br />")
Next
%>
```

ASP Application 对象

在一起协同工作以完成某项任务的一组 **ASP** 文件称作应用程序（**application**）。**ASP** 中的 **Application** 对象用于将这些文件捆绑在一起。

Application 对象

web 上的一个应用程序可以是一组 ASP 文件。这些 ASP 文件一起协同工作来完成某项任务。ASP 中的 Application 对象用来把这些文件捆绑在一起。

Application 对象用于存储和访问来自任何页面的变量，类似于 session 对象。不同之处在于，所有的用户分享一个 Application 对象，而 session 对象和用户的关系是一一对应的。

Application 对象存有会被应用程序中的许多页面使用的信息（比如数据库连接信息）。这意味着可以从任何的页面访问这些信息。同时也意味着你可在一个地点改变这些信息，然后这些改变会自动反映在所有的页面上。

存储和取回 Application 变量

Application 变量可被应用程序中的任何页面访问和改变。

可以像这样在 "Global.asa" 中创建 Application 变量：

```
<script language="vbscript" runat="server">

Sub Application_OnStart
application("vartime")=""
application("users")=1
End Sub

</script>
```

在上面的例子中，我们创建了两个 Application 变量："vartime" 和 "users"。

可以像这样访问 Application 变量的值：

```
There are
<%
Response.Write(Application("users"))
%>
active connections.
```

遍历 Contents 集合

Contents 集合包含着所有的 application 变量。我们可以通过对 contents 集合进行遍历，来查看其中存储的变量：

```
<%
dim i
For Each i in Application.Contents
  Response.Write(i & "<br />")
Next
%>
```

如果你不清楚 contents 集中的项目数量，可使用 count 属性：

```
<%
dim i
dim j
j=Application.Contents.Count
For i=1 to j
  Response.Write(Application.Contents(i) & "<br />")
Next
%>
```

遍历 StaticObjects 集合

可通过循环 StaticObjects 集合，来查看所有存储于 Application 对象中的对象的值：

```
<%
dim i
For Each i in Application.StaticObjects
  Response.Write(i & "<br />")
Next
%>
```

锁定和解锁

我们可以使用 "Lock" 方法来锁定应用程序。当应用程序锁定后，用户们就无法改变 Application 变量了（除了正在访问 Application 变量的用户）。我们也可使用 "Unlock" 方法来对应用程序进行解锁。这个方法会移除对 Application 变量的锁定：

```
<%
Application.Lock
  'do some application object operations
Application.Unlock
%>
```

ASP 文件引用

#include 指令用于在多重页面上创建需重复使用的函数、页眉、页脚或者其他元素等。

#include 指令

通过使用 #include 指令，我们可以在服务器执行 ASP 文件之前，把另一个ASP文件插入这个文件中。#include 命令用于在多个页面上创建需要重复使用的函数、页眉、页脚或者其他元素等。

如何使用 #include 指令

这里有一个名为 "mypage.asp" 的文件:

```
<html>
<body>
<h2>Words of Wisdom:</h2>
<p><!--#include file="wisdom.inc"--></p>
<h2>The time is:</h2>
<p><!--#include file="time.inc"--></p>
</body>
</html>
```

这是 "wisdom.inc" 文件:

```
"One should never increase, beyond what is necessary,
the number of entities required to explain anything."
```

这是 "time.inc" 文件:

```
<%
Response.Write(Time)
%>
```

在浏览器中查看的源代码应该类似这样：

```
<html>
<body>
<h2>Words of Wisdom:</h2>
<p>"One should never increase, beyond what is necessary,
the number of entities required to explain anything."</p>
<h2>The time is:</h2>
<p>11:33:42 AM</p>
</body>
</html>
```

Including 文件的语法：

如需在 ASP 中引用文件，请把 #include 命令置于注释标签之中：

```
<!--#include virtual="somefilename"-->
```

或者：

```
<!--#include file ="somefilename"-->
```

关键词 Virtual

关键词 virtual 指示路径以虚拟目录开始。

如果 "header.inc" 文件位于虚拟目录 /html 中，下面这行代码会插入文件 "header.inc" 中的内容：

```
<!-- #include virtual ="/html/header.inc" -->
```

关键词 File

关键词 File 指示一个相对的路径。相对路径起始于含有引用文件的目录。

假设文件位于 html 文件夹的子文件夹 headers 中，下面这段代码可引用 "header.inc" 文件的内容：

```
<!-- #include file ="headers\header.inc" -->
```

注意：被引用文件的路径是相对于引用文件的。假如包含 #include 声明的文件不在 html 目录中，这个声明就不会起效。

您也可以使用关键词 file 和语法 (..) 来引用上级目录中的文件。

提示和注释

在上面的一节中，我们使用 ".inc" 来作为被引用文件的后缀。注意：假如用户尝试直接浏览 INC 文件，这个文件中内容就会暴露。假如被引用的文件中的内容涉及机密，那么最好还是使用 "asp" 作为后缀。ASP 文件中的源代码被编译后是不可见的。被引用的文件也可引用其他文件，同时一个 ASP 文件可以对同一个文件引用多次。

重要事项：在脚本执行前，被引用的文件就会被处理和插入。

下面的代码无法执行，这是由于 ASP 会在为变量赋值之前执行 #include 命令：

```
<%
fname="header.inc"
%>
<!--#include file="<%=fname%>"-->
```

不能在脚本分隔符之间包含文件引用：

```
<%
For i = 1 To n
  <!--#include file="count.inc"-->
Next
%>
```

但是这段脚本可以工作：

```
<% For i = 1 to n %>
<!--#include file="count.inc" -->
<% Next %>
```

ASP Global.asa 文件

Global.asa 文件是一个可选的文件，它可包含可被 **ASP** 应用程序中每个页面访问的对象、变量以及方法的声明。

Global.asa 文件

Global.asa 文件是一个可选的文件，它可包含可被 ASP 应用程序中每个页面访问的对象、变量以及方法的声明。所有合法的浏览器脚本都能在 Global.asa 中使用。

Global.asa 文件可包含下列内容：

- Application 事件
- Session 事件
- <object> 声明
- TypeLibrary 声明
- ## include 指令

注释：Global.asa 文件须存放于 ASP 应用程序的根目录中，且每个应用程序只能有一个 Global.asa 文件。

Global.asa 中的事件

在 Global.asa 中，我们可以告知 application 和 session 对象在启动和结束时做什么事情。完成这项任务的代码被放置在事件操作器中。Global.asa 文件能包含四种类型的事件：

Application_OnStart - 此事件会在首位用户从 ASP 应用程序调用第一个页面时发生。此事件会在 web 服务器重起或者 Global.asa 文件被编辑之后发生。"Session_OnStart" 事件会在此事件发生之后立即发生。

Session_OnStart - 此事件会在每当新用户请求他或她的在 ASP 应用程序中的首个页面时发生。

Session_OnEnd - 此事件会在每当用户结束 session 时发生。在规定的时间（默认的事件为 20 分钟）内如果没有页面被请求，session 就会结束。

Application_OnEnd - 此事件会在最后一位用户结束其 session 之后发生。典型的情况是，此事件会在 Web 服务器停止时发生。此子程序用于在应用程序停止后清除设置，比如删除记录或者向文本文件写信息。

Global.asa 文件可能类似这样：

```
<script language="vbscript" runat="server">

sub Application_OnStart
  'some code
end sub

sub Application_OnEnd
  'some code
end sub

sub Session_OnStart
  'some code
end sub

sub Session_OnEnd
  'some code
end sub

</script>
```

注释：由于无法使用 ASP 的脚本分隔符 (<% 和 %>) 在 Global.asa 文件中插入脚本，我们需使用 HTML 的 <script> 元素。

<object> 声明

可通过使用 <object> 标签在 Global.asa 文件中创建带有 session 或者 application 作用域的对象。

注释：<object> 标签应位于 <script> 标签之外。

语法：

```
<object runat="server" scope="scope" id="id"
{progid="progID"|classid="classID"}>
....
</object>
```

参数	描述
scope	设置对象的作用域（作用范围）（Session 或者 Application）。
id	为对象指定一个唯一的 id。
ProgID	与 ClassID 关联的 id。ProgID 的格式是：[Vendor.]Component[.Version]。ProgID 或 ClassID 必需被指定。
ClassID	为 COM 类对象指定唯一的 id。ProgID 或 ClassID 必需被指定。

实例

第一个实例创建了一个名为 "MyAd" 且使用 ProgID 参数的 session 作用域对象：

```
<object runat="server" scope="session" id="MyAd" progid="MSWC.AdRotator">
</object>
```

第二个实例创建了名为 "MyConnection" 且使用 ClassID 参数的

```
<object runat="server" scope="application" id="MyConnection"
classid="Clsid:8AD3067A-B3FC-11CF-A560-00A0C9081C21">
</object>
```

在此 Global.asa 文件中声明的这些对象可被应用程序中的任何脚本使用。

GLOBAL.ASA:

```
<object runat="server" scope="session" id="MyAd" progid="MSWC.AdRotator">
</object>
```

您可以从 ASP 应用程序中的任意页面引用此 "MyAd" 对象：

某个 .ASP 文件:

```
<%=MyAd.GetAdvertisement("/banners/adrot.txt")%>
```

TypeLibrary 声明

TypeLibrary （类型库）是一个容器，其中装有对应于 COM 对象的 DLL 文件。通过在 Global.asa 中包含对 TypeLibrary 的调用，可以访问 COM 对象的常量，同时 ASP 代码也能更好地报告错误。假如您的站点的应用程序依赖于已在类型库中声明过数据类型的 COM 对象，您可以在 Global.asa 中对类型库进行声明。

语法：

```
<!--METADATA TYPE="TypeLib"
file="filename"
uuid="typelibraryuuid"
version="versionnumber"
lcid="localeid"
-->
```

参数	描述
file	规定指向类型库的绝对路径。参数 file 或者 uuid，两者缺一不可。
uuid	规定了针对类型库的唯一的标识符。参数 file 或者 uuid，两者缺一不可。
version	可选。用于选择版本。假如没有找到指定的版本，将使用最接近的版本。
lcid	可选。用于类型库的地区标识符。

错误值

服务器会返回以下的错误消息之一：

错误	代码	描述
ASP	0222	Invalid type library specification
ASP	0223	Type library not found
ASP	0224	Type library cannot be loaded
ASP	0225	Type library cannot be wrapped

注释：METADATA 标签可位于 Global.asa 文件中的任何位置（在 <script> 标签的内外均可）。不过，我们还是推荐将 METADATA 标签放置于 Global.asa 文件的顶部。

限定

关于可以在 Global.asa 文件中引用的内容的限定：

你无法显示 Global.asa 文件中的文本。此文件无法显示信息。

你只能在 Application_OnStart 和 Application_OnEnd 子例程中使用 Server 和 Application 对象。在 Session_OnEnd 子例程中，你可以使用 Server、Application 和 Session 对象。在 Session_OnStart 子例程中，你可使用任何内建的对象。

如何使用子例程

Global.asa 常用于对变量进行初始化。

下面的例子展示如何检测访问者首次到达站点的确切时间。时间存储在名为 "started" 的 Session 对象中，并且 "started" 变量的值可被应用程序中的任何 ASP 页面访问：

```
<script language="vbscript" runat="server">
sub Session_OnStart
Session("started")=now()
end sub
</script>
```

Global.asa 也可用于控制页面访问。

下面的例子展示如何把每位新的访问者重定向到另一个页面，在这个例子中会定向到 "newpage.asp" 这个页面：

```
<script language="vbscript" runat="server">
sub Session_OnStart
Response.Redirect("newpage.asp")
end sub
</script>
```

我们还可以在 Global.asa 中包含函数。

在下面的例子中，当 Web 服务器启动时，Application_OnStart 子例程也会启动。随后，Application_OnStart 子例程会调用另一个名为 "getcustomers" 的子例程。"getcustomers" 子例程会打开一个数据库，然后从 "customers" 表中取回一个记录集。此记录集会赋值给一个数组，在不查询数据库的情况下，任何 ASP 页面都能够访问这个数组：

```
<script language="vbscript" runat="server">

sub Application_OnStart
getcustomers
end sub

sub getcustomers
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=conn.execute("select name from customers")
Application("customers")=rs.GetRows
rs.Close
conn.Close
end sub

</script>
```

Global.asa 实例

在这个例子中，我们要创建一个可计算当前访客的 Global.asa 文件。

Application_OnStart 设置当服务器启动时，Application 变量 "visitors" 的值为 0。

每当有新用户访问时，Session_OnStart 子例程就会给变量 "visitors" 加 1。

每当 Session_OnEnd 子例程被触发时，此子例程就会从变量 "visitors" 减 1。

Global.asa 文件：

```
<script language="vbscript" runat="server">

Sub Application_OnStart
Application("visitors")=0
End Sub

Sub Session_OnStart
Application.Lock
Application("visitors")=Application("visitors")+1
Application.UnLock
End Sub

Sub Session_OnEnd
Application.Lock
Application("visitors")=Application("visitors")-1
Application.UnLock
End Sub

</script>
```

此 ASP 文件会显示当前用户的数目：

```
<html>
<head>
</head>
<body>
<p>There are <%response.write(Application("visitors"))%> online now!</p>
</body>
</html>
```

ASP 使用 CDOSYS 发送电子邮件

CDOSYS 是 **ASP** 中的内置组件。此组建用于通过 **ASP** 来发送电子邮件。

使用 CDOSYS 发送电子邮件

CDO (Collaboration Data Objects) 是一项微软的技术，设计目的是用来简化通信程序的创建。

CDOSYS 是 ASP 中的内置组件。我们会向您展示如何使用该组件来发送电子邮件。

CDONTs 怎么样？

微软已经在 Windows 2000、Windows XP 以及 Windows 2003 中淘汰了 CDONTs。如果您还在应用程序中使用 CDONTs，就需要更新代码，并使用新的 CDO 技术。

使用 CDOSYS 的实例

发送电子邮件：

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.TextBody="This is a message."
myMail.Send
set myMail=nothing
%>
```

发送带有 Bcc 和 CC 字段的文本邮件:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.Bcc="someoneelse@somedomain.com"
myMail.Cc="someoneelse2@somedomain.com"
myMail.TextBody="This is a message."
myMail.Send
set myMail=nothing
%>
```

发送 HTML 邮件:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.HTMLBody = "<h1>This is a message.</h1>"
myMail.Send
set myMail=nothing
%>
```

发送一封发送来自网站的网页的 HTML 邮件：

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.CreateMHTMLBody "http://www.w3school.com.cn/asp/"
myMail.Send
set myMail=nothing
%>
```

发送一封发送来自电脑中文件的网页的 HTML 邮件：

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.CreateMHTMLBody "file://c:/mydocuments/test.htm"
myMail.Send
set myMail=nothing
%>
```

发送一封带有附件的电子邮件：

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.TextBody="This is a message."
myMail.AddAttachment "c:\mydocuments\test.txt"
myMail.Send
set myMail=nothing
%>
```

使用远程服务器发送一封文本邮件：

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.TextBody="This is a message."
myMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/sendusing")=2
'远程 SMTP 服务器的 IP 或名称
myMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpserver") _
="smtp.server.com"
'服务器端口
myMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpserverport") _
=25
myMail.Configuration.Fields.Update
myMail.Send
set myMail=nothing
%>
```

ASP 高级

ASP Response 对象

ASP Response 对象用于从服务器向用户发送输出的结果。

实例

使用 ASP 写文本

本例演示如何使用 ASP 来写文本。

```
<html>
<body>

<%
response.write("Hello World!")
%>

</body>
</html>
```

在 ASP 中使用 HTML 标签格式化文本

本例演示如何使用 ASP 将文本和 HTML 标签结合起来。

```
<html>
<body>
<%
response.write("<h2>您可以使用 HTML 标签来格式化文本</h2>")
%>

<%
response.write("<p style='color:#0000ff'>这段文本的样式是通过 style 属性添加的。</p>")
%>
</body>
</html>
```

将用户重定向至不同的 URL

本例演示如何将用户重定向至另一个的 URL。

```
<%
if Request.Form("select")<>"" then
       Response.Redirect(Request.Form("select"))
end if
%>

<html>
<body>

<form action="/example/aspe/demo_aspe_redirect.asp" method="post">

<input type="radio" name="select"
value="/example/aspe/demo_aspe_server.asp">
服务器实例<br />

<input type="radio" name="select"
value="/example/aspe/demo_aspe_text.asp">
文本实例<br /><br />
<input type="submit" value="跳转！">

</form>

</body>
</html>
```

显示随机的链接

本例演示一个超级链接，当您每次载入页面时，它将显示两个链接中的其中一个。

```
<html>
<body>

<%
randomize()
r=rnd()
if r>0.5 then
  response.write("<a href='http://www.w3school.com.cn'>W3School.com.cn!</a>")
else
  response.write("<a href='http://www.news.cn'>news.cn!</a>")
end if
%>

<p>
本例演示一个链接，每当您加载本页时，就会显示两个链接之一：W3School.com.cn! 或 news.cn！各占百分之五十的
</p>

</body>
</html>
```

控制缓存

本例演示如何控制缓存。

```
<%
Response.Buffer=true
%>
<html>
<body>
<p>
当您的 response 缓存清空时, 这段文本就会发送到浏览器。
</p>
<%
Response.Flush
%>
</body>
</html>
```

清空缓存

本例演示如何清空缓存。

```
<%
Response.Buffer=true
%>
<html>
<body>
<p>这是我希望发送给用户的文本。</p>
<p>不, 我改变主意了。我希望清除这些文本。</p>
<%
Response.Clear
%>
</body>
</html>
```

在处理过程中终止脚本并返回结果

本例演示如何在处理过程中中断脚本的运行。

```
<html>
<body>
<p>我正在写文本。这些文本不会被<br />
<%
Response.End
%>
完全发送。这时候已经不能输出任何文本了！</p>
</body>
</html>
```

设置在页面失效前把页面在浏览器中缓存多少分钟

本例演示如何规定页面在失效前在浏览器中的缓存时间。

```
<%Response.Expires=-1%>
<html>
<body>
<p>每当被访问, 本页都会被刷新！</p>
</body>
</html>
```

设置页面缓存在浏览器中的失效日期或时间

本例演示如何规定页面在浏览器中的缓存时间日期或时间

```
<%
Response.ExpiresAbsolute=#May 05,2001 05:30:30#
%>
<html>
<body>
<p>本页面的缓存会在该日期失效：05, 2001 05:30:30！</p>
</body>
</html>
```

检查用户是否仍然与服务器相连

本例演示如何检查用户是否已与服务器断开。

```
<html>
<body>

<%
If Response.IsClientConnected=true then
Response.Write("用户仍然保持连接。")
else
Response.Write("用户未连接。")
end if
%>

</body>
</html>
```

设置内容类型

本例演示如何规定内容的类型。

```
<%
Response.ContentType="text/html"
%>
<html>
<body>

<p>This is some text.</p>

</body>
</html>
```

设置字符集

本例演示如何规定字符集的名称。

```
<%
Response.Charset="ISO8859-1"
%>
<html>
<body>

<p>This is some text</p>

</body>
</html>
```

Response 对象

ASP Response 对象用于从服务器向用户发送输出的结果。它的集合、属性和方法如下：

集合

集合	描述
Cookies	设置 cookie 的值。假如不存在，就创建 cookie ，然后设置指定的值。

属性

属性	描述
Buffer	规定是否缓存页面的输出。
CacheControl	设置代理服务器是否可以缓存由 ASP 产生的输出。
Charset	将字符集的名称追加到 Response 对象中的 content-type 报头。
ContentType	设置 Response 对象的 HTTP 内容类型。
Expires	设置页面在失效前的浏览器缓存时间（分钟）。
ExpiresAbsolute	设置浏览器上页面缓存失效的日期和时间。
IsClientConnected	指示客户端是否已从服务器断开。
Pics	向 response 报头的 PICS 标志追加值。
Status	规定由服务器返回的状态行的值。

方法

方法	描述
AddHeader	向 HTTP 响应添加新的 HTTP 报头和值。
AppendToLog	向服务器记录项目（server log entry）的末端添加字符串。
BinaryWrite	在没有任何字符转换的情况下直接向输出写数据。
Clear	清除已缓存的 HTML 输出。
End	停止处理脚本，并返回当前的结果。
Flush	立即发送已缓存的 HTML 输出。
Redirect	把用户重定向到另一个 URL。
Write	向输出写指定的字符串。

ASP Request 对象

ASP Request 对象用于从用户那里取得信息。

实例

QueryString 集合 实例

当用户点击链接时发送查询信息

本例演示如何在链接中向页面发送一些额外的查询信息，并在目标页面中取回这些信息（在本例中是同一页面）。

```
<html>
<body>

<a href="/example/aspe/demo_aspe_simplequerystring.asp?color=green">Example</a>

<%
Response.Write(Request.QueryString)
%>

</body>
</html>
```

对 QueryString 集合的简单应用

本例演示 *QueryString* 集合如何从表单取回值。此表单使用 GET 方法，这意外着所发送的信息对用户来说是可见的（在地址中）。GET 方法还会限制所发送信息的数量。

```
<html>
<body>

<form action="/example/aspe/demo_aspe_simplereqquery.asp" method="get">
First name: <input type="text" name="fname"><br />
Last name: <input type="text" name="lname"><br />
<input type="submit" value="Submit">
</form>

<%
Response.Write(Request.QueryString)
%>

</body>
</html>
```

如何使用从表单传来的信息

本例演示如何使用从表单取回的值。我们会使用 *QueryString* 集合。此表单使用 GET 方法。

```
<html>
<body>
<form action="/example/aspe/demo_aspe_reqquery.asp" method="get">
您的姓名：<input type="text" name="fname" size="20" />
<input type="submit" value="提交" />
</form>
<%
dim fname
fname=Request.QueryString("fname")
If fname<>"" Then
      Response.Write("你好！" & fname & "！<br />")
      Response.Write("今天过得怎么样？")
End If
%>
</body>
</html>
```

来自表单的更多信息

本例演示假如输入字段包含若干相同的名称的话，*QueryString* 会包含什么内容。它将展示如何把这些相同的名称分隔开来。它也会展示如何使用 *count* 关键词来对 "name" 属性进行计数。此表单使用 GET 方法。

```
<html>
<body>

<%
If Request.QueryString<>"" Then
      If Request.QueryString("name")<>", " Then
           name1=Request.QueryString("name")(1)
           name2=Request.QueryString("name")(2)
      end if
end if
%>

<form action="/example/aspe/demo_aspe_reqquery2.asp" method="get">
First name:
<input type="text" name="name" value="<%=name1%>" />
<br />
Last name:
<input type="text" name="name" value="<%=name2%>" />
<br />
<input type="submit" value="Submit" />
</form>
<hr>
<%
If Request.QueryString<>"" Then
      Response.Write("<p>")
      Response.Write("The information received from the form was:")
      Response.Write("</p><p>")
      Response.Write("name=" & Request.QueryString("name"))
      Response.Write("</p><p>")
      Response.Write("The name property's count is: ")
      Response.Write(Request.QueryString("name").Count)
      Response.Write("</p><p>")
      Response.Write("First name=" & name1)
      Response.Write("</p><p>")
      Response.Write("Last name=" & name2)
      Response.Write("</p>")
end if
%>
</body>
</html>
```

Form 集合 实例

一个 Form 集合的简单应用

本例演示 *Form* 集合如何从表单取回值。此表单使用 POST 方法，这意味着发送的信息对用户来说是不可见的，并且对所发送信息的量没有限制（可发送大量的信息）。

```
<html>
<body>

<form action="/example/aspe/demo_aspe_simpleform1.asp" method="post">
First name:
<input type="text" name="fname" value="Donald" />
<br />
Last name:
<input type="text" name="lname" value="Duck" />
<br />
<input type="submit" value="Submit" />
</form>

<%
Response.Write(Request.Form)
%>

</body>
</html>
```

如何使用来自表单的信息

本例演示如何使用从表单取回的信息。我们使用了 *Form* 集合。表单使用了 POST 方法。

```
<html>
<body>
<form action="/example/aspe/demo_aspe_simpleform.asp" method="post">
您的姓名：<input type="text" name="fname" size="20" />
<input type="submit" value="提交" />
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>"" Then
      Response.Write("您好！" & fname & "！<br />")
      Response.Write("今天过得怎么样？")
End If
%>
</body>
</html>
```

来自表单的更多信息

本例演示假如若干的输入域使用了相同的名称，*Form* 集合会包含什么信息。它将展示如何把这些相同的名称分割开来。它也会展示如何使用 *count* 关键词来对 "name" 属性进行计数。此表单使用 POST 方法。

```
<html>
<body>

<form action="/example/aspe/demo_aspe_form2.asp" method="post">
First name:
<input type="text" name="name" value="Donald" />
<br />
Last name:
<input type="text" name="name" value="Duck" />
<br />
<input type="submit" value="Submit" />
</form>
<hr />

<p>来自上面的表单的信息：</p>
<%
If Request.Form("name")<>"" Then
      Response.Write("<p>")
      Response.Write("name=" & Request.Form("name"))
      Response.Write("</p><p>")
      Response.Write("name 属性的数目：")
      Response.Write(Request.Form("name").Count)
      Response.Write("</p><p>")
      Response.Write("First name=" & Request.Form("name")(1))
      Response.Write("</p><p>")
      Response.Write("Last name=" & Request.Form("name")(2))
      Response.Write("</p>")
End if
%>

</body>
</html>
```

带有单选按钮的表单

本例演示如何使用 *Form* 集合通过单选按钮与用户进行交互。此表单使用 POST 方法。

```
<html>
<%
dim cars
cars=Request.Form("cars")
%>
<body>
<form action="/example/aspe/demo_aspe_radiob.asp" method="post">
<p>请选择您喜欢的汽车：</p>

<input type="radio" name="cars"
<%if cars="Volvo" then Response.Write("checked")%>
value="Volvo">Volvo</input>
<br />
<input type="radio" name="cars"
<%if cars="Saab" then Response.Write("checked")%>
value="Saab">Saab</input>
<br />
<input type="radio" name="cars"
<%if cars="BMW" then Response.Write("checked")%>
value="BMW">BMW</input>
<br /><br />
<input type="submit" value="提交" />
</form>
<%
if cars<>"" then
   Response.Write("<p>您喜欢的汽车是" & cars & "</p>")
end if
%>
</body>
</html>
```

带有复选按钮的表单

本例演示如何使用 *Form* 集合通过复选按钮与用户进行交互。此表单使用 POST 方法。

```
<html>
<body>
<%
fruits=Request.Form("fruits")
%>

<form action="/example/aspe/demo_aspe_checkboxes.asp" method="post">
<p>您喜欢哪些水果：</p>
<input type="checkbox" name="fruits" value="Apples"
<%if instr(fruits,"Apple") then Response.Write("checked")%>>
Apple
<br />
<input type="checkbox" name="fruits" value="Oranges"
<%if instr(fruits,"Oranges") then Response.Write("checked")%>>
Orange
<br />
<input type="checkbox" name="fruits" value="Bananas"
<%if instr(fruits,"Banana") then Response.Write("checked")%>>
Banana
<br />
<input type="submit" value="提交">
</form>
<%
if fruits<>"" then%>
   <p>您喜欢：<%Response.Write(fruits)%></p>
<%end if
%>

</body>
</html>
```

其他实例

获取用户信息

如何查明访问者的浏览器类型、IP 地址等信息。

```
<html>
<body>
<p>
<b>您正在通过这款浏览器访问我们的站点：</b>
<%Response.Write(Request.ServerVariables("http_user_agent"))%>
</p>
<p>
<b>您的 IP 地址是：</b>
<%Response.Write(Request.ServerVariables("remote_addr"))%>
</p>
<p>
<b>IP 地址的 DNS 查询是：</b>
<%Response.Write(Request.ServerVariables("remote_host"))%>
</p>
<p>
<b>调用该页面所用的方法是：</b>
<%Response.Write(Request.ServerVariables("request_method"))%>
</p>
<p>
<b>服务器的域名：</b>
<%Response.Write(Request.ServerVariables("server_name"))%>
</p>
<p>
<b>服务器的端口：</b>
<%Response.Write(Request.ServerVariables("server_port"))%>
</p>
<p>
<b>服务器的软件：</b>
<%Response.Write(Request.ServerVariables("server_software"))%>
</p>

</body>
</html>
```

获取服务器变量

本例演示如何使用 *ServerVariables* 集合取得访问者的浏览器类型、IP 地址等信息。

```
<html>
<body>

<p>
所有可能的服务器变量：
</p>
<%
For Each Item in Request.ServerVariables
      Response.Write(Item & "<br />")
Next
%>

</body>
</html>
```

创建 welcome cookie

本例演示如何使用 Cookies 集合创建一个欢迎 *cookie*。

```
<%
dim numvisits
response.cookies("NumVisits").Expires=date+365
numvisits=request.cookies("NumVisits")

if numvisits="" then
   response.cookies("NumVisits")=1
   response.write("欢迎！这是您第一次访问本页面。")
else
   response.cookies("NumVisits")=numvisits+1
   response.write("之前，您已经访问过本页面 ")
   response.write(numvisits & " 次。")
end if
%>
<html>
<body>
</body>
</html>
```

探测用户发送的字节总数

本例演示如何使用 *TotalBytes* 属性来取得用户在 Request 对象中发送的字节总数。

```
<html>
<body>

<form action="/example/aspe/demo_aspe_totalbytes.asp" method="post">
请键入一些字符：
<input type="text" name="txt"><br /><br />
<input type="submit" value="提交">
</form>

<%
If Request.Form("txt")<>"" Then
   Response.Write("您提交了：")
   Response.Write(Request.Form)
   Response.Write("<br /><br />")
   Response.Write("字节总计：")
   Response.Write(Request.Totalbytes)
End If
%>

</body>
</html>
```

Request 对象

当浏览器向服务器请求页面时，这个行为就被称为一个 request（请求）。

ASP Request 对象用于从用户那里获取信息。它的集合、属性和方法描述如下：

集合

集合	描述
ClientCertificate	包含了在客户证书中存储的字段值
Cookies	包含了 HTTP 请求中发送的所有 cookie 值
Form	包含了使用 post 方法由表单发送的所有的表单（输入）值
QueryString	包含了 HTTP 查询字符串中所有的变量值
ServerVariables	包含了所有的服务器变量值

属性

属性	描述
TotalBytes	返回在请求正文中客户端所发送的字节总数

方法

方法	描述
BinaryRead	取回作为 post 请求的一部分而从客户端送往服务器的数据，并把它存放到一个安全的数组之中。

ASP Application 对象

在一起协同工作以完成某项任务的一组 **ASP** 文件称为一个应用程序。而 **ASP** 中的 **Application** 对象的作用是把这些文件捆绑在一起。

Application 对象

Web 上的一个应用程序可以是一组 ASP 文件。这些 ASP 在一起协同工作来完成一项任务。而 ASP 中的 Application 对象的作用是把这些文件捆绑在一起。

Application 对象用于存储和访问来自任意页面的变量，类似 Session 对象。不同之处在于所有的用户分享一个 Application 对象，而 session 对象和用户的关系是一一对应的。

Application 对象掌握的信息会被应用程序中的很多页面使用（比如数据库连接信息）。这就意味我们可以从任意页面访问这些信息。也意味着你可以在在一个页面上改变这些信息，随后这些改变会自动地反映到所有的页面中。

Application 对象的集合、方法和事件的描述如下：

集合

集合	描述
Contents	包含所有通过脚本命令追加到应用程序中的项目。
StaticObjects	包含所有使用 HTML 的 <object> 标签追加到应用程序中的对象。

方法

方法	描述
Contents.Remove	从 Contents 集合中删除一个项目。
Contents.RemoveAll()	从 Contents 集合中删除所有的项目。
Lock	防止其余的用户修改 Application 对象中的变量。
Unlock	使其他的用户可以修改 Application 对象中的变量（在被 Lock 方法锁定之后）。

事件

事件	描述
Application_OnEnd	当所有用户的 session 都结束，并且应用程序结束时，此事件发生。
Application_OnStart	在首个新的 session 被创建之前（这时 Application 对象被首次引用），此事件会发生。

ASP Session 对象

Session 对象用于存储关于某个用户会话（**session**）的信息，或者修改相关的设置。存储在 **session** 对象中的变量掌握着单一用户的信息，同时这些信息对于页面中的所有页面都是可用的。

实例

设置并返回 LCID

本例演示 "LCID" 属性。此属性设置并返回一个指示位置或者地区的整数。类似于日期、时间以及货币等内容都要根据位置或者地区来显示。

```
<html>
<body>

<%
response.write("<p>")
response.write("本页的默认 LCID 是：" & Session.LCID & "<br />")
response.write("上面的 LCID 的日期格式是：" & date() & "<br />")
response.write("上面的 LCID 的货币格式是：" & FormatCurrency(350))
response.write("</p>")

Session.LCID=1036

response.write("<p>")
response.write("现在 LCID 变更为：" & Session.LCID & "<br />")
response.write("上面的 LCID 的日期格式是：" & date() & "<br />")
response.write("上面的 LCID 的货币格式是：" & FormatCurrency(350))
response.write("</p>")

Session.LCID = 3079

response.write("<p>")
response.write("现在 LCID 变更为：" & Session.LCID & "<br />")
response.write("上面的 LCID 的日期格式是：" & date() & "<br />")
response.write("上面的 LCID 的货币格式是：" & FormatCurrency(350))
response.write("</p>")

Session.LCID = 2057

response.write("<p>")
response.write("现在 LCID 变更为：" & Session.LCID & "<br />")
response.write("上面的 LCID 的日期格式是：" & date() & "<br />")
response.write("上面的 LCID 的货币格式是：" & FormatCurrency(350))
response.write("</p>")
%>

</body>
</html>
```

返回 SessionID

本例演示 "SessionID" 属性。该属性为每位用户返回一个唯一的 id。这个 id 由服务器生成。

```
<html>
<body>

<%
Response.Write(Session.SessionID)
%>

</body>
</html>
```

session 的超时

本例演示 "Timeout" 属性。这个例子设置并返回 session 的超时时间（分钟）。

```
<html>
<body>

<%
response.write("<p>")
response.write("默认 Timeout 是：" & Session.Timeout & " 分钟。")
response.write("</p>")

Session.Timeout=30

response.write("<p>")
response.write("现在的 Timeout 是 " & Session.Timeout & " 分钟。")
response.write("</p>")
%>

</body>
</html>
```

Session 对象

当您正在操作一个应用程序时，您会启动它，然后做些改变，随后关闭它。这个过程很像一次对话（Session）。计算机知道你是谁。它也知道你在何时启动和关闭这个应用程序。但是在因特网上，问题出现了：web 服务器不知道你是谁，也不知道你做什么，这是由于 HTTP 地址无法留存状态（信息）。

ASP 通过为每个用户创一个唯一的 cookie 解决了这个问题。cookie 发送到服务器，它包含了可识别用户的信息。这个接口称作 Session 对象。

Session 对象用于存储关于某个用户会话（session）的信息，或者修改相关的设置。存储在 session 对象中的变量掌握着单一用户的信息，同时这些信息对于页面中的所有页面都是可用的。存储于 session 变量中的信息通常是 name、id 以及参数等。服务器会为每位新用户创建一个新的 Session 对象，并在 session 到期后撤销这个对象。

下面是 Session 对象的集合、属性、方法以及事件：

集合

集合	描述
Contents	包含所有通过脚本命令追加到 session 的条目。
StaticObjects	包含了所有使用 HTML 的 <object> 标签追加到 session 的对象。

属性

属性	描述
CodePage	规定显示动态内容时使用的字符集
LCID	设置或返回指定位置或者地区的一个整数。诸如日期、时间以及货币的内容会根据位置或者地区来显示。
SessionID	为每个用户返回一个唯一的 id。此 id 由服务器生成。
Timeout	设置或返回应用程序中的 session 对象的超时时间（分钟）。

方法

方法	描述
Abandon	撤销一个用户的 session。
Contents.Remove	从 Contents 集合删除一个项目。
Contents.RemoveAll()	从 Contents 集合删除全部项目。

事件

事件	描述
Session_OnEnd	当一个会话结束时此事件发生。
Session_OnStart	当一个会话开始时此事件发生。

ASP Server 对象

ASP Server 对象的作用是访问有关服务器的属性和方法。

实例

此文件最后被修改的时间是？

探测文件的最后更新时间。

```
<html>
<body>

<%
Set fs = Server.CreateObject("Scripting.FileSystemObject")
Set rs = fs.GetFile(Server.MapPath("/example/aspe/demo_aspe_lastmodified.asp"))
modified = rs.DateLastModified
%>
本文件的最后修改时间是：<%response.write(modified)
Set rs = Nothing
Set fs = Nothing
%>

</body>
</html>
```

打开并读取某个文本文件

本例会打开文件 "Textfile.txt" 以供读取。

```
<html>
<body>

<%
Set FS = Server.CreateObject("Scripting.FileSystemObject")
Set RS = FS.OpenTextFile(Server.MapPath("/example/aspe") & "\textfile.txt",1)
While not rs.AtEndOfStream
      Response.Write RS.ReadLine
      Response.Write("<br />")
Wend
%>

<p>
<a href="/example/aspe/textfile.txt">查看此文本文件</a>
</p>

</body>
</html>
```

自制的点击计数器

本例可从一个文件中读取一个数字，并在此数字上累加 1，然后将此数写回这个文件。

```
<%
Set FS=Server.CreateObject("Scripting.FileSystemObject")
Set RS=FS.OpenTextFile(Server.MapPath("/example/aspe/counter.txt"), 1, False)
fcount=RS.ReadLine
RS.Close

fcount=fcount+1

'This code is disabled due to the write access security on our server:
'Set RS=FS.OpenTextFile(Server.MapPath("counter.txt"), 2, False)
'RS.Write fcount
'RS.Close

Set RS=Nothing
Set FS=Nothing

%>
<html>
<body>
<p>
本页已被访问了 <%=fcount%>  次。
</p>
</body>
</html>
```

Server 对象

ASP Server 对象的作用是访问有关服务器的属性和方法。其属性和方法描述如下：

属性

属性	描述
ScriptTimeout	设置或返回在一段脚本终止前它所能运行时间（秒）的最大值。

方法

方法	描述
CreateObject	创建对象的实例（instance）。
Execute	从另一个 ASP 文件中执行一个 ASP 文件。
GetLastError()	返回可描述已发生错误状态的 ASPError 对象。
HTMLEncode	将 HTML 编码应用到某个指定的字符串。
MapPath	将一个指定的地址映射到一个物理地址。
Transfer	把一个 ASP 文件中创建的所有信息传输到另一个 ASP 文件。
URLEncode	把 URL 编码规则应用到指定的字符串。

ASP ASPError 对象

ASPError 对象用于显示在 **ASP** 文件的脚本中发生的任何错误的详细信息。

ASP ASPError 对象

ASP 3.0 提供这个对象，且在 IIS5 及更高版本中可用。

ASPError 对象用于显示在 ASP 文件的脚本中发生的任何错误的详细信息。当 Server.GetLastError 被调用时，ASPError 对象就会被创建，因此只能通过使用 Server.GetLastError 方法来访问错误信息。

ASPError 对象的属性描述如下（所有属性都是可读的）：

注释：下面的属性只能 Server.GetLastError() 方法来访问。

属性

属性	描述
ASPCode	返回由 IIS 生成的错误代码。
ASPDescription	返回有关错误的详细信息。（假如错误和 ASP 相关。）
Category	返回错误来源。(是由 ASP、脚本语言还是对象引起的？)
Column	返回在出错文件中的列位置。
Description	返回关于错误的简短描述。
File	返回出错 ASP 文件的文件名。
Line	返回错误所在的行数。
Number	返回关于错误的标准 COM 错误代码。
Source	返回错误所在行的实际的源代码

ASP FileSystemObject 对象

FileSystemObject 对象用于访问服务器上的文件系统。

实例

指定的文件存在吗？

本例演示如何首先创建 FileSystemObject 对象，然后使用 FileExists 方法来探测某文件是否存在。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

If (fs.FileExists("c:\windows\cursors\xxx.cur"))=true Then

Response.Write("文件 c:\windows\cursors\xxx.cur 存在。")

Else
      Response.Write("文件 c:\windows\cursors\xxx.cur 不存在。")
End If

set fs=nothing
%>

</body>
</html>
```

指定的文件夹存在吗？

本例演示如何使用 FolderExists 方法探测某文件夹是否存在。

```
<html>
<body>
<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

If fs.FolderExists("c:\temp") = true Then
      Response.Write("文件夹 c:\temp 存在。")
Else
      Response.Write("文件夹 c:\temp 不存在。")
End If

set fs=nothing
%>

</body>
</html>
```

指定的驱动器存在吗？

本例演示如何使用 DriveExists 方法来探测某个驱动器是否存在。

```
<html>
<body>
<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

if fs.driveexists("c:") = true then
      Response.Write("驱动器 c: 存在。")
Else
      Response.Write("驱动器 c: 不存在。")
End If

Response.write("<br />")

if fs.driveexists("g:") = true then
      Response.Write("驱动器 g: 存在。")
Else
      Response.Write("驱动器 g: 不存在。")
End If

set fs=nothing
%>

</body>
</html>
```

取得某个指定驱动器的名称

本例演示如何使用 GetDriveName 方法来取得某个指定的驱动器的名称。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")
p=fs.GetDriveName("c:\windows\cursors\abc.cur")

Response.Write("驱动器名称是：" & p)

set fs=nothing
%>

</body>
</html>
```

取得某个指定路径的父文件夹的名称

本例演示如何使用 GetParentFolderName 方法来取得某个指定的路径的父文件夹的名称。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")
p=fs.GetParentFolderName("c:\winnt\cursors\3dgarro.cur")

Response.Write("c:\windows\cursors\abc.cur 的父文件夹名称是：" & p)

set fs=nothing
%>

</body>
</html>
```

取得文件夹扩展名

本例演示如何使用 GetExtensionName 方法来取得指定的路径中的最后一个成分的文件扩展名。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")
p=fs.GetParentFolderName("c:\winnt\cursors\3dgarro.cur")

Response.Write("c:\windows\cursors\abc.cur 的父文件夹名称是：" & p)

set fs=nothing
%>

</body>
</html>
```

取得文件名

本例演示如何使用 GetFileName 方法来取得指定的路径中的最后一个成分的文件名。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Response.Write("这个文件名的最后一个成分是：")
Response.Write(fs.GetFileName("c:\windows\cursors\abc.cur"))
set fs=nothing
%>

</body>
</html>
```

取得文件或文件夹的基名称

本例演示如何使用 GetBaseName 方法来返回在指定的路径中文件或者文件夹的基名称。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Response.Write(fs.GetBaseName("c:\windows\cursors\abc.cur"))
Response.Write("<br />")
Response.Write(fs.GetBaseName("c:\windows\cursors\"))
Response.Write("<br />")
Response.Write(fs.GetBaseName("c:\windows\"))

set fs=nothing
%>

</body>
</html>
```

FileSystemObject 对象

FileSystemObject 对象用于访问服务器上的文件系统。此对象可对文件、文件夹以及目录路径进行操作。也可通过此对象获取文件系统的信息。

下面的代码会创建一个文本文件 (c:\test.txt)，然后向这个文件写一些文本：

```
<%
dim fs,fname
      set fs=Server.CreateObject("Scripting.FileSystemObject")
      set fname=fs.CreateTextFile("c:\test.txt",true)
      fname.WriteLine("Hello World!")
      fname.Close
set fname=nothing
set fs=nothing
%>
```

FileSystemObject 对象的属性和方法描述如下：

属性

属性	描述
Drives	返回本地计算机上所有驱动器对象的集合。

方法

方法	描述
BuildPath	将一个名称追加到已有的路径后
CopyFile	从一个位置向另一个位置拷贝一个或多个文件。
CopyFolder	从一个位置向另一个位置拷贝一个或多个文件夹。
CreateFolder	创建新文件夹。
CreateTextFile	创建文本文件，并返回一个 TextStream 对象。
DeleteFile	删除一个或者多个指定的文件。
DeleteFolder	删除一个或者多个指定的文件夹。
DriveExists	检查指定的驱动器是否存在。
FileExists	检查指定的文件是否存在。
FolderExists	检查某个文件夹是否存在。
GetAbsolutePathName	针对指定的路径返回从驱动器根部起始的完整路径。
GetBaseName	返回指定文件或者文件夹的基名称。
GetDrive	返回指定路径中所对应的驱动器的 Drive 对象。
GetDriveName	返回指定的路径的驱动器名称。
GetExtensionName	返回在指定的路径中最后一个成分的文件扩展名。
GetFile	返回一个针对指定路径的 File 对象。
GetFileName	返回在指定的路径中最后一个成分的文件名。
GetFolder	返回一个针对指定路径的 Folder 对象。
GetParentFolderName	返回在指定的路径中最后一个成分的父文件名称。
GetSpecialFolder	返回某些 Windows 的特殊文件夹的路径。
GetTempName	返回一个随机生成的文件或文件夹。
MoveFile	从一个位置向另一个位置移动一个或多个文件。
MoveFolder	从一个位置向另一个位置移动一个或多个文件夹。
OpenTextFile	打开文件，并返回一个用于访问此文件的 TextStream 对象。

ASP TextStream 对象

TextStream 对象用于访问文本文件的内容。

实例

读文件

本例演示如何使用 FileSystemObject 的 OpenTextFile 方法来创建一个 TextStream 对象。TextStream 对象的 ReadAll 方法会从已打开的文本文件中取得内容。

```
<html>
<body>
<p>这就是文本文件中的文本：</p>
<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Set f=fs.OpenTextFile(Server.MapPath("/example/aspe/testread.txt"), 1)
Response.Write(f.ReadAll)
f.Close

Set f=Nothing
Set fs=Nothing
%>
</body>
</html>
```

读文本文件中的一个部分

本例演示如何仅仅读取一个文本流文件的部分内容。

```
<html>
<body>
<p>这是从文本文件中读取的前 5 个字符：</p>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)
Response.Write(f.Read(5))
f.Close

Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

读文本文件中的一行

本例演示如何从一个文本流文件中读取一行内容。

```
<html>
<body>
<p>这是从文本文件中读取的第一行：</p>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)
Response.Write(f.ReadLine)
f.Close

Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

读取文本文件的所有行

本例演示如何从文本流文件中读取所有的行。

```
<html>
<body>
<p>这是从文本文件中读取的所有行：</p>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)

do while f.AtEndOfStream = false
Response.Write(f.ReadLine)
Response.Write("<br />")
loop

f.Close
Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

略过文本文件的一部分

本例演示如何在读取文本流文件时跳过指定的字符数。

```
<html>
<body>
<p>文本文件中的前 4 个字符被略掉了：</p>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)
f.Skip(4)
Response.Write(f.ReadAll)
f.Close

Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

略过文本文件的一行

本例演示如何在读取文本流文件时跳过一行。

```
<html>
<body>
<p>文本文件中的第一行被略掉了：</p>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)
f.SkipLine
Response.Write(f.ReadAll)
f.Close

Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

返回行数

本例演示如何返回在文本流文件中的当前行号。

```
<html>
<body>
<p>这是文本文件中的所有行（带有行号）：</p>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)

do while f.AtEndOfStream = false
Response.Write("Line:" & f.Line & " ")
Response.Write(f.ReadLine)
Response.Write("<br />")
loop

f.Close
Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

取得列数

本例演示如何取得在文件中当前字符的列号。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)
Response.Write(f.Read(2))
Response.Write("<p>指针目前位于文本文件中的位置 " & f.Column & "。</p>")
f.Close

Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

TextStream 对象

TextStream 对象用于访问文本文件的内容。

下面的代码会创建一个文本文件 (c:\test.txt)，然后向此文件写一些文本（变量 f 是 TextStream 对象的一个实例）：

```
<%
dim fs, f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.CreateTextFile("c:\test.txt",true)
f.WriteLine("Hello World!")
f.Close
set f=nothing
set fs=nothing
%>
```

如需创建TextStream对象的一个实例，我们可以使用 FileSystemObject 对象的 CreateTextFile 方法或者 OpenTextFile 方法，也可以使用 File 对象的 OpenAsTextStream 方法。

TextStream 对象的属性和方法描述如下：

属性

属性	描述
AtEndOfLine	在 TextStream 文件中，如果文件指针正好位于行尾标记的前面，那么该属性值返回 True；否则返回 False。
AtEndOfStream	如果文件指针在 TextStream 文件末尾，则该属性值返回 True；否则返回 False。
Column	返回 TextStream 文件中当前字符位置的列号。
Line	返回 TextStream 文件中的当前行号。

方法

方法	描述
Close	关闭一个打开的 TextStream 文件。
Read	从一个 TextStream 文件中读取指定数量的字符并返回结果（得到的字符串）。
ReadAll	读取整个 TextStream 文件并返回结果。
ReadLine	从一个 TextStream 文件读取一整行（到换行符但不包括换行符）并返回结果。
Skip	当读一个 TextStream 文件时跳过指定数量的字符。
SkipLine	当读一个 TextStream 文件时跳过下一行。
Write	写一段指定的文本（字符串）到一个 TextStream 文件。
WriteLine	写入一段指定的文本（字符串）和换行符到一个 TextStream 文件中。
WriteBlankLines	写入指定数量的换行符到一个 TextStream 文件中。

ASP Drive 对象

Drive 对象用于返回关于本地磁盘驱动器或者网络共享驱动器的信息。

实例

取得指定驱动器的可用空间数

本例演示如何首先创建一个 FileSystemObject 对象，然后使用 AvailableSpace 属性来获得指定驱动器的可用空间。

```
<html>
<body>

<%
Set fs=Server.CreateObject("Scripting.FileSystemObject")

Set f=fs.OpenTextFile(Server.MapPath("testread.txt"), 1)
Response.Write(f.Read(2))
Response.Write("<p>指针目前位于文本文件中的位置 " & f.Column & "。</p>")
f.Close

Set f=Nothing
Set fs=Nothing
%>

</body>
</html>
```

取得指定驱动器的剩余空间容量

本例演示如何使用 FreeSpace 空间属性来取得指定驱动器的剩余空间。

```
<html>
<body>

<%
Dim fs, d, n
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set d=fs.GetDrive("c:")
n = "驱动器：" & d
n = n & "<br />以字节计的剩余空间：" & d.FreeSpace
Response.Write(n)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

取得指定驱动器的总容量

本例演示如何使用 TotalSize 属性来获得指定驱动器的总容量。

```
<html>
<body>

<%
Dim fs,d,n
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set d=fs.GetDrive("c:")
n = "驱动器：" & d
n = n & "<br />以字节计的总容量：" & d.TotalSize
Response.Write(n)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

取得指定驱动器的驱动器字母

本例演示如何使用 DriveLetter 属性来获得指定驱动器的驱动器字母。

```
<html>
<body>

<%
dim fs, d, n
set fs=Server.CreateObject("Scripting.FileSystemObject")
set d=fs.GetDrive("c:")
Response.Write("驱动器字母是：" & d.driveletter)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

取得指定驱动器的驱动器类型

本例演示如何使用 DriveType 属性来获得指定驱动器的驱动器类型。

```
<html>
<body>

<%
dim fs, d, n
set fs=Server.CreateObject("Scripting.FileSystemObject")
set d=fs.GetDrive("c:")
Response.Write("驱动器类型是：" & d.DriveType)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

取得指定驱动器的文件系统信息

本例演示如何使用 FileSystem 来取得指定驱动器的文件系统类型。

```
<html>
<body>

<%
dim fs, d, n
set fs=Server.CreateObject("Scripting.FileSystemObject")
set d=fs.GetDrive("c:")
Response.Write("文件系统是：" & d.FileSystem)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

驱动器是否已就绪？

本例演示如何使用 IsReady 属性来检查指定的驱动器是否已就绪。

```
<html>
<body>

<%
dim fs,d,n
set fs=Server.CreateObject("Scripting.FileSystemObject")
set d=fs.GetDrive("c:")
n = "此 " & d.DriveLetter
if d.IsReady=true then
    n = n & " 驱动器已就绪。"
else
    n = n & " 驱动器未就绪。"
end if
Response.Write(n)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

取得指定驱动器的路径

本例演示如何使用 Path 属性来取得指定驱动器的路径。

```

<html>
<body>

<%
dim fs,d
set fs=Server.CreateObject("Scripting.FileSystemObject")
set d=fs.GetDrive("c:")
Response.Write("路径是：" & d.Path)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

取得指定驱动器的根文件夹

本例演示如何使用 RootFolder 属性来取得指定驱动器的根文件夹。

```
<html>
<body>

<%
dim fs,d
set fs=Server.CreateObject("Scripting.FileSystemObject")
set d=fs.GetDrive("c:")
Response.Write("根文件是：" & d.RootFolder)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

取得指定驱动器的序列号

本例演示如何使用 Serialnumber 属性来取得指定驱动器的序列号。

```
<html>
<body>

<%
dim fs,d
set fs=Server.CreateObject("Scripting.FileSystemObject")
set d=fs.GetDrive("c:")
Response.Write("序列号：" & d.SerialNumber)
set d=nothing
set fs=nothing
%>

</body>
</html>
```

Drive 对象

Drive 对象用于返回关于本地磁盘驱动器或者网络共享驱动器的信息。Drive 对象可以返回有关驱动器的文件系统、剩余容量、序列号、卷标名等信息。

注释：无法通过 Drive 对象返回有关驱动器内容的信息。要达到这个目的，请使用 Folder 对象。

如需操作 Drive 对象的相关属性，我们需要创建通过 FileSystemObject 对象来创建 Drive 对象的实例。首先，创建一个 FileSystemObject 对象，然后通过 FileSystemObject 对象的 GetDrive 方法或者 Drives 属性来例示 Drive 对象。

下面的例子使用 FileSystemObject 对象的 GetDrive 方法来例示 Drive 对象，并使用 TotalSize 属性来返回指定驱动器 (c:) 的容量总数（字节）：

```
<%
Dim fs,d
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set d=fs.GetDrive("c:")
Response.Write("Drive " & d & ":")
Response.Write("Total size in bytes: " & d.TotalSize)
set d=nothing
set fs=nothing
%>
```

输出：

Drive c: Total size in bytes: 5893563398

Drive 对象的属性

属性	描述
AvailableSpace	向用户返回在指定的驱动器或网络共享驱动器上的可用空间容量。
DriveLetter	返回识别本地驱动器或网络共享驱动器的大写字母。
DriveType	返回指定驱动器的类型。
FileSystem	返回指定驱动器所使用的文件系统类型。
FreeSpace	向用户返回在指定的驱动器或网络共享驱动器上的剩余空间容量。
IsReady	如果指定驱动器已就绪，则返回 true。否则返回 false。
Path	返回其后有一个冒号的大写字母，用来指示指定驱动器的路径名。
RootFolder	返回一个文件夹对象，该文件夹代表指定驱动器的根文件夹。
SerialNumber	返回指定驱动器的序列号。
ShareName	返回指定驱动器的网络共享名。
TotalSize	返回指定的驱动器或网络共享驱动器的总容量
VolumeName	设置或者返回指定驱动器的卷标名

ASP File 对象

File 对象用于返回关于指定文件的信息。

实例

文件何时被创建？

本例演示如何首先创建 FileSystemObject 对象，然后使用 File 对象的 DateCreated 属性来取得指定文件被创建的日期和时间。

```
<html>
<body>

<%
dim fs, f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.GetFile(Server.MapPath("testread.txt"))
Response.Write("文件 testread.txt 的创建时间是：" & f.DateCreated)
set f=nothing
set fs=nothing
%>

</body>
</html>
```

此文件何时被修改？

本例演示如何使用 DateLastModified 属性来取得指定文件被修改的日期和时间。

```
<html>
<body>

<%
dim fs, f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.GetFile(Server.MapPath("testread.txt"))
Response.Write("文件 testread.txt 的最后修改时间是：" & f.DateLastModified)
set f=nothing
set fs=nothing
%>

</body>
</html>
```

此文件何时被访问过？

此例演示如何使用 DateLastAccessed 属性来取得指定文件最后被访问的日期和时间。

```
<html>
<body>

<%
dim fs, f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.GetFile(Server.MapPath("testread.txt"))
Response.Write("文件 testread.txt 的最后访问时间是：" & f.DateLastAccessed)
set f=nothing
set fs=nothing
%>

</body>
</html>
```

返回指定文件的属性

本例演示如何使用 Attributes 来返回指定文件的属性。

```
<html>
<body>

<%
dim fs,f
set fs=Server.CreateObject("Scripting.FileSystemObject")
set f=fs.GetFile(Server.MapPath("testread.txt"))
Response.Write("文件 testread.txt 的属性是：" & f.Attributes)
set f=nothing
set fs=nothing
%>

</body>
</html>
```

File 对象

File 对象用于返回有关指定文件的信息。

如需操作 File 对象的相关属性和方法，我们需要通过 FileSystemObject 来创建 File 对象的实例。首先，创建一个 FileSystemObject 对象，然后通过 FileSystemObject 对象的 GetFile 方法，或者通过 Folder 对象的 Files 属性来例示此 File 对象。

下面的代码使用 FileSystemObject 对象的 GetFile 方法来例示这个 File 对象，并使用 DateCreated 属性来返回指定文件被创建的日期：

```
<%
Dim fs,f
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set f=fs.GetFile("c:\test.txt")
Response.Write("File created: " & f.DateCreated)
set f=nothing
set fs=nothing
%>
```

输出：

```
File created: 8/8/2008 10:01:19 AM
```

File 对象的属性和方法

属性

属性	描述
Attributes	设置或返回指定文件的属性。
DateCreated	返回指定文件创建的日期和时间。
DateLastAccessed	返回指定文件最后被访问的日期和时间。
DateLastModified	返回指定文件最后被修改的日期和时间。
Drive	返回指定文件或文件夹所在的驱动器的驱动器字母。
Name	设置或返回指定文件的名称。
ParentFolder	返回指定文件或文件夹的父文件夹对象。
Path	返回指定文件的路径。
ShortName	返回指定文件的短名称（8.3 命名约定）。
ShortPath	返回指定文件的短路径（8.3 命名约定）。
Size	返回指定文件的尺寸（字节）。
Type	返回指定文件的类型。

方法

方法	描述
Copy	把指定文件从一个位置拷贝到另一个位置。
Delete	删除指定文件。
Move	把指定文件从一个位置移动到另一个位置。
OpenAsTextStream	打开指定文件，并返回一个 TextStream 对象以便访问此文件。

ASP Folder 对象

Folder 对象用来返回有关指定文件夹的信息。

Folder 对象

Folder 对象用于返回有关指定文件夹的信息。

如需操作 Folder 对象，我们需要通过 FileSystemObject 对象来创建 Folder 对象的实例。首先，创建一个 FileSystemObject 对象，然后通过 FileSystemObject 对象的 GetFolder 方法来例示这个 Folder 对象。

下面的代码使用 FileSystemObject 对象的 GetFolder 方法来例示这个 Folder 对象，并使用 DateCreated 属性来返回指定文件的创建日期：

```
<%
Dim fs,fo
Set fs=Server.CreateObject("Scripting.FileSystemObject")
Set fo=fs.GetFolder("c:\test")
Response.Write("Folder created: " & fo.DateCreated)
set fo=nothing
set fs=nothing
%>
```

输出：

```
Folder created: 10/22/2001 10:01:19 AM
```

Folder 对象的集合、属性以及方法

集合

集合	描述
Files	返回指定文件夹中所有文件夹的集合。
SubFolders	返回指定文件夹中所有子文件夹的集合。

属性

属性	描述
Attributes	设置或返回指定文件夹的属性。
DateCreated	返回指定文件夹被创建的日期和时间。
DateLastAccessed	返回指定文件夹最后被访问的日期和时间。
DateLastModified	返回指定文件夹最后被修改的日期和时间。
Drive	返回指定文件夹所在的驱动器的驱动器字母。
IsRootFolder	假如文件夹是根文件夹，则返回 ture，否则返回 false。
Name	设置或返回指定文件夹的名称。
ParentFolder	返回指定文件夹的父文件夹。
Path	返回指定文件的路径。
ShortName	返回指定文件夹的短名称。（8.3 命名约定）
ShortPath	返回指定文件夹的短路径。（8.3 命名约定）
Size	返回指定文件夹的大小。
Type	返回指定文件夹的类型。

方法

方法	描述
Copy	把指定的文件夹从一个位置拷贝到另一个位置。
Delete	删除指定文件夹。
Move	把指定的文件夹从一个位置移动到另一个位置。
CreateTextFile	在指定的文件夹创建一个新的文本文件，并返回一个 TextStream 对象以访问这个文件。

ASP Dictionary 对象

Dictionary 对象用于在结对的名称/值中存储信息（等同于键和项目）。

实例

指定的键存在吗？

本例演示如何受首先创建一个 Dictionary 对象，然后使用 Exists 方法来检查指定的键是否存在。

```
<html>
<body>

<%
dim d
set d=Server.CreateObject("Scripting.Dictionary")
d.Add "c", "China"
d.Add "i", "Italy"
if d.Exists("c")= true then
    Response.Write("键存在。")
else
    Response.Write("键不存在。")
end if
set d=nothing
%>

</body>
</html>
```

返回一个所有项目的数组

本例演示如何使用 Items 方法来返回所有项目的一个数组。

```
<html>
<body>

<%
dim d,a,i,s
set d=Server.CreateObject("Scripting.Dictionary")
d.Add "c", "China"
d.Add "i", "Italy"

Response.Write("<p>项目的值是：</p>")
a=d.Items
for i = 0 To d.Count -1
    s = s & a(i) & "<br />"
next
Response.Write(s)

set d=nothing
%>

</body>
</html>
```

返回一个所有键的数组

本例演示如何使用 Keys 方法来返回所有键的一个数组。

```
<html>
<body>

<%
dim d,a,i,s
set d=Server.CreateObject("Scripting.Dictionary")
d.Add "c", "China"
d.Add "i", "Italy"
Response.Write("<p>键的值是：</p>")
a=d.Keys
for i = 0 To d.Count -1
    s = s & a(i) & "<br />"
next
Response.Write(s)
set d=nothing
%>

</body>
</html>
```

返回某个项目的值

本例演示如何使用 Item 属性来返回一个项目的值。

```
<html>
<body>

<%
dim d
set d=Server.CreateObject("Scripting.Dictionary")
d.Add "c", "China"
d.Add "i", "Italy"
Response.Write("项目 i 的值是：" & d.item("i"))
set d=nothing
%>

</body>
</html>
```

设置一个键

本例演示如何使用 Key 属性来在 Dictionary 对象中设置一个键。

```
<html>
<body>

<%
dim d
set d=Server.CreateObject("Scripting.Dictionary")
d.Add "c", "China"
d.Add "i", "Italy"
d.Key("i") = "it"
Response.Write("键 i 已设置为 it，其值是：" & d.Item("it"))
set d=nothing
%>

</body>
</html>
```

返回键/项目对的数目

本例演示如何使用 Count 属性来返回键/项目对的数目。

```
<html>
<body>

<%
dim d, a, s, i
set d=Server.CreateObject("Scripting.Dictionary")
d.Add "c", "China"
d.Add "i", "Italy"
Response.Write("key/item 对的数目是：" & d.Count)
set d=nothing
%>

</body>
</html>
```

Dictionary 对象

Dictionary 对象用于在结对的名称/值中存储信息（(等同于键和项目）。Dictionary 对象看似比数组更为简单，然而，Dictionary 对象却是更令人满意的处理关联数据的解决方案。

比较 Dictionary 和数组：

- 键用于识别 Dictionary 对象中的项目
- 无需调用 ReDim 来改变 Dictionary 对象的尺寸
- 当从 Dictionary 删除一个项目时，其余的项目会自动上移
- Dictionary 不是多维，而数组是
- Dictionary 与数组相比，有更多的内建对象
- Dictionary 在频繁地访问随机元素时，比数组工作得更好
- Dictionary 在根据它们的内容定位项目时，比数组工作得更好

下面的例子创建了一个 Dictionary 对象，并向对象添加了一些键/项目对，然后取回了键 bl 的值：

```
<%
Dim d
Set d=Server.CreateObject("Scripting.Dictionary")
d.Add "re","Red"
d.Add "gr","Green"
d.Add "bl","Blue"
d.Add "pi","Pink"
Response.Write("The value of key bl is: " & d.Item("bl"))
%>
```

输出：

```
The value of key bl is: Blue
```

Dictionary 对象的属性和方法描述如下：

属性

属性	描述
CompareMode	设置或返回用于在 Dictionary 对象中比较键的比较模式。
Count	返回 Dictionary 对象中键/项目对的数目。
Item	设置或返回 Dictionary 对象中一个项目的值。
Key	为 Dictionary 对象中已有的键值设置新的键值。

方法

方法	描述
Add	向 Dictionary 对象添加新的键/项目对。
Exists	返回一个逻辑值，这个值可指示指定的键是否存在于 Dictionary 对象中。
Items	返回 Dictionary 对象中所有项目的一个数组。
Keys	返回 Dictionary 对象中所有键的一个数组。
Remove	从 Dictionary 对象中删除指定的键/项目对。
RemoveAll	删除 Dictionary 对象中所有的键/项目对。

ADO 简介

ADO 用于从网页访问数据库。

从 ASP 页面访问数据库

从 ASP 文件内部访问数据库的通常途径是：

1. 创建至数据库的 ADO 连接（ADO connection）
2. 打开数据库连接
3. 创建 ADO 记录集（ADO recordset）
4. 打开记录集（recordset）
5. 从数据集中提取你所需要的数据
6. 关闭数据集
7. 关闭连接

什么是 ADO?

- ADO 是一项微软公司的技术
- ADO 指 ActiveX Data Objects
- ADO 是一个微软的 Active-X 组件
- ADO 会随着微软 IIS 自动安装
- ADO 是用以访问数据库中数据的编程接口

下一步学习什么内容？

假如您希望学习更多关于 ADO 的知识，请阅读我们的 ADO 教程。

ASP 组件

ASP AdRotator 组件

实例

简单的 AdRotator 实例

本例展示：每当用户访问网站或者刷新一次页面，如何使用 AdRotator 组件来显示一幅不同的广告图像。

```
<html>
<body>

<%
set adrotator=Server.CreateObject("MSWC.AdRotator")
adrotator.Border="2"
Response.Write(adrotator.GetAdvertisement("/example/aspe/advertisements.txt"))
%>

<p>
<b>注释：</b>由于图像是随机变化的，
同时由于本页可供选择的图片很少，
因此经常会出现两次显示同一广告的情况。
</p>

<p>
<a href="/example/aspe/advertisements.txt">
查看 advertisements.txt
</a>
</p>

</body>
</html>
```

AdRotator - 图片链接

本例展示：每当用户访问网站或者刷新一次页面，如何使用 AdRotator 组件来显示一幅不同的广告图像。此外，图片本身就是链接。

```
<%
url=Request.QueryString("url")
If url<>"" then Response.Redirect(url)
%>
<html>
<body>

<%
set adrotator=Server.CreateObject("MSWC.AdRotator")
adrotator.TargetFrame="target='_blank'"
response.write(adrotator.GetAdvertisement("/example/aspe/advertisements2.txt"))
%>

<p>
<b>注释：</b>由于图像是随机变化的，
同时由于本页可供选择的图片很少，
因此经常会出现两次显示同一广告的情况。
</p>

<p>
<a href="/example/aspe/advertisements2.txt">
查看 advertisements2.txt
</a>
</p>
</body>
</html>
```

ASP AdRotator 组件

每当用户进入网站或刷新页面时，ASP AdRotator 组件就会创建一个 AdRotator 对象来显示一幅不同的图片。

语法：

```
<%
set adrotator=server.createobject("MSWC.AdRotator")
adrotator.GetAdvertisement("textfile.txt")
%>
```

实例

假设我们有一个文件名为 "banners.asp"。它类似于这样：

```
<html%>
<body%>
<%
set adrotator=Server.CreateObject("MSWC.AdRotator")
response.write(adrotator.GetAdvertisement("ads.txt"))
%>
</body%>
</html%>
```

文件 "ads.txt" 类似这样：

```
*
w3school.gif
http://www.w3school.com.cn/
Visit W3School
80
microsoft.gif
http://www.microsoft.com/
Visit Microsoft
20
```

"ads.txt" 文件中星号下面的代码定义了如何显示这些图像，链接地址，图像的替换文本，在每百次点击中的显示几率。我们可以看到，W3School 图片的显示几率是 80%，而 Microsoft 图片的显示几率是 20%。

注释：为了使这些链接在用户点击时可以正常工作，我们需要对文件 "ads.txt" 进行一点点小小的修改：

```
REDIRECT banners.asp
*
w3school.gif
http://www.w3school.com.cn/
Visit W3School
80
microsoft.gif
http://www.microsoft.com/
Visit Microsoft
20
```

转向页面会接收到名为 url 的变量的查询字符串，其中含有供转向的 URL。

注释：如需规定图像的高度、宽度和边框，我们可以在 REDIRECT 下面插入这些代码：

```
REDIRECT banners.asp
WIDTH 468
HEIGHT 60
BORDER 0
*
w3school.gif
...
...
```

最后要做的是把这些代码加入文件"banners.asp"中：

```
<%
url=Request.QueryString("url")
If url<>"" then Response.Redirect(url)
%>
<html>
<body>
<%
set adrotator=Server.CreateObject("MSWC.AdRotator")
response.write(adrotator.GetAdvertisement("textfile.txt"))
%>
</body>
</html>
```

好了，这就是全部的内容！

AdRotator 组件的属性

Border 属性

规定围绕广告的边框的尺寸。

```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
adrot.Border="2"
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

Clickable 属性

规定广告本身是否是超级链接。

```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
adrot.Clickable=false
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

TargetFrame 属性

显示广告的框架名称。

```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
adrot.TargetFrame="target='_blank'"
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

AdRotator 组件的方法

GetAdvertisement 方法

返回在页面中显示广告的 HTML。

```
<%
set adrot=Server.CreateObject("MSWC.AdRotator")
Response.Write(adrot.GetAdvertisement("ads.txt"))
%>
```

ASP Browser Capabilities 组件

实例

Browser Capabilities 组件

本例演示如何测定每一个访问网站的浏览器的类型、性能以及版本号。

```
<html>
<body>

<%
Set MyBrow=Server.CreateObject("MSWC.BrowserType")
%>

<table border="1" width="65%">
   <tr>
     <td width="52%">客户机操作系统</td>
     <td width="48%"><%=MyBrow.platform%></td>
   </tr>
   <tr>
     <td >Web 浏览器</td>
     <td ><%=MyBrow.browser%></td>
   </tr>
   <tr>
     <td>浏览器版本</td>
     <td><%=MyBrow.version%></td>
   </tr>
   <tr>
     <td>框架支持</td>
     <td><%=MyBrow.frames%></td>
   </tr>
   <tr>
     <td>表格支持</td>
     <td><%=MyBrow.tables%></td>
   </tr>
   <tr>
     <td>音频支持</td>
     <td><%=MyBrow.backgroundsounds%></td>
   </tr>
   <tr>
     <td>Cookies 支持</td>
     <td><%=MyBrow.cookies%></td>
   </tr>
   <tr>
     <td>VBScript 支持</td>
     <td><%=MyBrow.vbscript%></td>
   </tr>
   <tr>
     <td>JavaScript 支持</td>
     <td><%=MyBrow.javascript%></td>
   </tr>
</table>

</body>
</html>
```

ASP Browser Capabilities 组件

ASP Browser Capabilities 组件会创建一个 BrowserType 对象，这个对象可测定访问者浏览器的类型、性能以及版本号。

当浏览器连接到服务器时，就会向服务器发送一个 HTTP User Agent 报头。这个报头包含着有关浏览器的信息（比如浏览器类型和版本号）。BrowserType 对象会把报头中的信息与服务器上名为 "Browscap.ini" 的文件中的信息作比较。

如果标题中被发送的浏览器类型和版本号和 "Browsercap.ini" 文件中信息可以匹配，那么我们就可以使用 BrowserType 对象列出这个匹配的浏览器的相关属性。如果上述情况不匹配，这个对象会把每个属性设置为 "UNKNOWN"。

语法

```
<%
Set MyBrow=Server.CreateObject("MSWC.BrowserType")
%>
```

下面的例子对在 ASP 文件中创建一个 BrowserType 对象，并显示一个展示当前浏览器性能的表格：

```
<html>
<body>

<%
Set MyBrow=Server.CreateObject("MSWC.BrowserType")
%>

<table border="1" width="100%">
<tr>
<th>Client OS</th>
<th><%=MyBrow.platform%></th>
</tr><tr>
<td >Web Browser</td>
<td ><%=MyBrow.browser%></td>
</tr><tr>
<td>Browser version</td>
<td><%=MyBrow.version%></td>
</tr><tr>
<td>Frame support?</td>
<td><%=MyBrow.frames%></td>
</tr><tr>
<td>Table support?</td>
<td><%=MyBrow.tables%></td>
</tr><tr>
<td>Sound support?</td>
<td><%=MyBrow.backgroundsounds%></td>
</tr><tr>
<td>Cookies support?</td>
<td><%=MyBrow.cookies%></td>
</tr><tr>
<td>VBScript support?</td>
<td><%=MyBrow.vbscript%></td>
</tr><tr>
<td>JavaScript support?</td>
<td><%=MyBrow.javascript%></td>
</tr>
</table>

</body>
</html>
```

输出：

Client OS	WinNT
Web Browser	IE
Browser version	5.0
Frame support?	True
Table support?	True
Sound support?	True
Cookies support?	True
VBScript support?	True
JavaScript support?	True

Browscap.ini文件

"Browsercap.ini" 文件用于声明属性，并设置各浏览器的默认值。

本节内容不是关于如何 Browscap.ini 文件的教程，我们只提供一些关于 "Browsercap.ini" 的基础知识和概念。

"Browsercap.ini" 文件可包含下面的信息：

```
[;comments]
[HTTPUserAgentHeader]
[parent=browserDefinition]
[property1=value1]
[propertyN=valueN]
[Default Browser Capability Settings]
[defaultProperty1=defaultValue1]
[defaultPropertyN=defaultValueN]
```

参数	描述
comments	可选项。任何起始于分号的代码行都被 BrowserType 对象忽略
HTTPUserAgentHeader	可选项。规定与在 propertyN 中设定的 browser-property 值声明相关的 HTTP User Agent 报头。允许使用通配符。
browserDefinition	可选项。规定作为父浏览器使用的某个浏览器的 HTTP User Agent header-string。当前浏览器的定义会继承在父浏览器的定义中所有声明过的属性值。
propertyN	可选项。规定浏览器的属性。下面的表格列出了某些可能的属性： `ActiveXControls` - 是否支持ActiveX控件? `Backgroundsounds` - 是否支持背景声音? `Cdf` - 是否支持针对网络广播（Webcasting）的频道定义格式（Channel Definition Format）？ `Tables` - 是否支持表格? `Cookies` - 是否支持cookies? `Frames` - 是否支持框架? `Javaapplets` - 是否支持Java applets? `Javascript` - 是否支持JScript? `Vbscript` - 是否支持VBScript? `Browser` - 定义浏览器的名称 `Beta` - 浏览器是否为beta软件? `Platform` - 规定浏览器运行的平台 `Version` - 规定浏览器的版本号。
valueN	可选项。规定 propertyN 的值。可为字符串、整数（前缀为 #）或者逻辑值。
defaultPropertyN	可选项。规定浏览器属性的名称，假如已定义的 HTTPUserAgentHeader 值中没有值能与浏览器发送的 HTTP 用户代理报头相匹配，则为这个属性分配一个默认的值。
defaultValueN	Optional. 规定 defaultPropertyN 的值。可为字符串、整数（前缀为 #）或者逻辑值。

"Browsercap.ini"文件会类似这样：

```
;IE 5.0
[IE 5.0]
browser=IE
Version=5.0
majorver=#5
minorver=#0
frames=TRUE
tables=TRUE
cookies=TRUE
backgroundsounds=TRUE
vbscript=TRUE
javascript=TRUE
javaapplets=TRUE
ActiveXControls=TRUE
beta=False;DEFAULT BROWSER
[*]
browser=Default
frames=FALSE
tables=TRUE
cookies=FALSE
backgroundsounds=FALSE
vbscript=FALSE
javascript=FALSE
```

ASP Content Linking 组件

实例

Content Linking 组件

本例会构建一个内容列表。

```
<html>
<body>

<p>下面的例子构建了一个内容列表：</p>

<%
dim c
dim i
set nl=server.createobject("MSWC.Nextlink")
c = nl.GetListCount("/example/aspe/links.txt")
i = 1
%>
<ul>
<%do while (i <= c) %>
<li><a href="<%=nl.GetNthURL("/example/aspe/links.txt", i)%>">
<%=nl.GetNthDescription("/example/aspe/links.txt", i)%></a>
<%
i = (i + 1)
loop
%>
</ul>

<p>文本文件包含页面 URL 的列表和链接描述。
每行文本针对一个页面。
请注意，URL 和描述必须由 TAB 字符分隔。
</p>

<p>
<a href="/example/aspe/links.txt">查看 links.txt</a>
</p>

</body>
</html>
```

Content Linking 组件 2

本例使用 Content Linking 组件在一个文本文件所列的页面间进行导航。

```
<html>
<body>

<h1>这是页面 1</h1>

<%
Set nl=Server.CreateObject("MSWC.NextLink")
If (nl.GetListIndex("/example/aspe/links2.txt")>1) Then
%>
<a href="<%Response.Write(nl.GetPreviousURL("/example/aspe/links2.txt"))%>">
上一页
</a>
<%End If%>

<a href="<%Response.Write(nl.GetNextURL("/example/aspe/links2.txt"))%>">
下一页
</a>

<p>本例使用 Content Linking 组件
对文本文件中的 URL 进行导航。</p>

<p><a href="/example/aspe/links2.txt">查看 links2.txt</a></p>
</body>
</html>
```

ASP Content Linking 组件

ASP Content Linking 组件用于创建快捷便利的导航系统。

Content Linking 组件会返回一个 Nextlink 对象，这个对象用于容纳需要导航网页的一个列表。

语法

```
<%
Set nl=Server.CreateObject( "MSWC.NextLink" )
%>
```

首先，我们会创建文本文件 - "links.txt"。此文件包含需要导航的页面的相关信息。页面的排列顺序应该与它们的显示顺序相同，并包含对每个文件的描述（使用制表符来分隔文件名和描述信息）。

注释：如果你希望向列表添加文件信息，或者改变在列表中的页面顺序，那么你需要做的所有事情仅仅是修改这个文本文件而已！然后导航系统会自动地更新！

"links.txt":

```
asp_intro.asp ASP 简介
asp_syntax.asp ASP 语法
asp_variables.asp ASP 变量
asp_procedures.asp ASP 程序
```

请在上面列出的页面中放置这行代码：<!-- #include file="nlcode.inc"-->。这行代码会在 "links.txt" 中列出每个页面上引用下面这段代码，这样导航就可以工作了。

"nlcode.inc":

```
<%
'Use the Content Linking Component
'to navigate between the pages listed
'in links.txt

dim nl
Set nl=Server.CreateObject("MSWC.NextLink")
if (nl.GetListIndex("links.txt")>1) then
  Response.Write("<a href='" & nl.GetPreviousURL("links.txt"))
  Response.Write("'>Previous Page</a>")
end if
Response.Write("<a href='" & nl.GetNextURL("links.txt"))
Response.Write("'>Next Page</a>")
%>
```

ASP Content Linking 组件的方法

GetListCount 方法

返回内容链接列表文件中所列项目的数目：

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetListCount("links.txt")
Response.Write("There are ")
Response.Write(c)
Response.Write(" items in the list")
%>
```

输出：

There are 4 items in the list

GetListIndex 方法

返回在内容链接列表文件中当前文件的索引号。第一个条目的索引号是 1。假如当前页面不在列表文件中，则返回 0。

例子

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetListIndex("links.txt")
Response.Write("Item number ")
Response.Write(c)
%>
```

输出：

Item number 3

GetNextDescription 方法

返回在内容链接列表文件中所列的下一个条目的文本描述。假如在列表文件中没有找到当前文件，则列表中最后一个页面的文本描述。

例子

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetNextDescription("links.txt")
Response.Write("Next ")
Response.Write("description is: ")
Response.Write(c)
%>
```

输出：Next description is: ASP Variables

GetNextURL 方法

返回在内容链接列表文件中所列的下一个条目的 URL。假如在列表文件中没有找到当前文件，则列表中最后一个页面的 URL。

例子

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetNextURL("links.txt")
Response.Write("Next ")
Response.Write("URL is: ")
Response.Write(c)
%>
```

输出：Next URL is: asp_variables.asp

GetNthDescription 方法

返在内容链接列表文件中所列的第 N 个页面的描述信息。

例子

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetNthDescription("links.txt",3)
Response.Write("Third ")
Response.Write("description is: ")
Response.Write(c)
%>
```

输出：Third description is: ASP Variables

GetNthURL 方法

返在内容链接列表文件中所列的第 N 个页面的 URL。

例子

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetNthURL("links.txt",3)
Response.Write("Third ")
Response.Write("URL is: ")
Response.Write(c)
%>
```

输出：Third URL is: asp_variables.asp

GetPreviousDescription 方法

返回在内容链接列表文件中所列前一个条目的文本描述。假如在列表文件中没有找到当前文件，则列表中第一个页面的文本描述。

例子

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetPreviousDescription("links.txt")
Response.Write("Previous ")
Response.Write("description is: ")
Response.Write(c)
%>
```

输出：Previous description is: ASP Variables

GetPreviousURL 方法

返回在内容链接列表文件中所列前一个条目的 URL。假如在列表文件中没有找到当前文件，则列表中第一个页面的URL。

例子

```
<%
dim nl,c
Set nl=Server.CreateObject("MSWC.NextLink")
c=nl.GetPreviousURL("links.txt")
Response.Write("Previous ")
Response.Write("URL is: ")
Response.Write(c)
%>
```

输出：Previous URL is: asp_variables.asp

ASP Content Rotator (ASP 3.0)

实例

Content Rotator 组件

每当用户访问或者刷新页面时，该组件就会显示不同的 HTML 内容字符串。

```
<html>
<body>

<p><b>注释：</b>如果服务器未安装 ASP 3.0，则本例无法运行。</p>

<p>
<a href="/example/aspe/textads.asp">查看 textads.txt</a>
</p>

<%
set cr=server.createobject("MSWC.ContentRotator")
response.write(cr.ChooseContent("/example/aspe/textads.txt"))
%>

<p>
<b>注释：</b>由于文本文件中的内容字符串是随机改变的，
同时本页只有四条内容可供选择，
因此，有时页面会连续显示两次相同的内容。
</p>

</body>
</html>
```

ASP Content Rotator 组件

ASP Content Rotator 组件会创建一个 ContentRotator 对象，每当用户访问或者刷新某个页面时，该对象就会显示一段不同的 HTML 内容字符串。一个名为内容目录文件（Content Schedule File）的文本文件包含着有关内容字符串的信息。

内容字符串可包含 HTML 标签，这样你就可以显示 HTML 可呈现的任何内容：文本、图像、颜色或者超级链接。

语法

```
<%
Set cr=Server.CreateObject( "MSWC.ContentRotator" )
%>
```

每当某用户查看网页时，下面这个例子就会显示不同的内容。首先在站点根目录的子文件夹 text 中创建一个名为 "textads.txt" 的文件。

"textads.txt":

```
%% #1
This is a great day!!

%% #2
<h1>Smile</h1>

%% #3
<img src="smiley.gif">

%% #4
Here's a <a href="http://www.w3school.com.cn">link</a>
```

注意：在每个内容字符串起始位置的#号码。这个号码是一个可选的参数，用来 HTML 内容字符串的相对权重。在本例中，Content Rotator 有十分之一的几率显示第一个内容字符串，有十分之二的几率显示第二个内容字符串，有十分之三的几率显示第三个字符串，而第四个字符串为十分之四的几率。

然后，创建一个 ASP 文件，并插入下面的代码：

```
<html>
<body>

<%
set cr=server.createobject("MSWC.ContentRotator")
response.write(cr.ChooseContent("text/textads.txt"))
%>

</body>
</html>
```

ASP Content Rotator 组件的方法

`ChooseContent`

获取并显示某个内容字符串

```
&lt;%
    dim cr
    Set cr=Server.CreateObject("MSWC.ContentRotator")
    response.write(cr.ChooseContent("text/textads.txt"))
    %&gt;
```

`GetAllContent`

取回并显示文本文件中所有的内容字符串

```
&lt;%
    dim cr
    Set cr=Server.CreateObject("MSWC.ContentRotator")
    response.write(cr.GetAllContent("text/textads.txt"))
    %&gt;
```

AJAX 与 ASP

AJAX 简介

AJAX 旨在不重载整个页面的情况下对网页的某些部分进行更新。

AJAX 是什么？

AJAX = Asynchronous JavaScript and XML（异步 JavaScript 和 XML）。

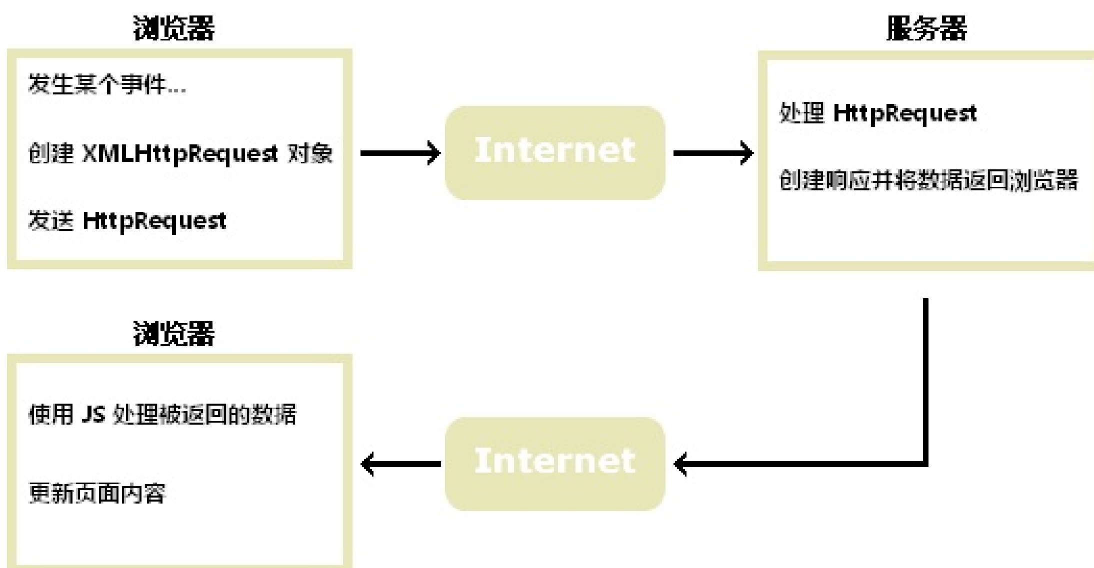
AJAX 是一种创建快速动态网页的技术。

通过在后台与服务器交换少量数据，AJAX 允许网页进行异步更新。这意味着，在不重新加载整个网页的情况下，对网页某些部分进行更新。

典型的网页（不使用 AJAX），如果内容发生变化，就必须重载整个页面。

使用 AJAX 的典型案例包括：谷歌地图、腾讯微博、优酷视频等等。

AJAX 如何工作



AJAX 基于因特网标准

AJAX 基于因特网标准，并使用以下技术组合：

- XMLHttpRequest 对象（与服务器异步交互数据）
- JavaScript/DOM （显示/取回信息）
- CSS （设置数据的样式）

- XML （常用作数据传输的格式）

提示：AJAX 应用程序是独立于浏览器和平台的！

谷歌搜索建议（Google Suggest）

随着谷歌搜索建议功能在 2005 的发布，AJAX 开始流行起来。

谷歌搜索建议使用 AJAX 创造出动态性极强的 web 界面：当您在谷歌的搜索框中键入内容时，JavaScript 会把字符发送到服务器，服务器则会返回建议列表。

今天就开始使用 AJAX

在我们的 ASP 教程中，我们将演示 AJAX 如何更新网页的某个部分，在不重载整个页面的情况下。我们将使用 ASP 来编写服务器端的脚步。

如果您希望学习更多有关 AJAX 的知识，请访问我们的 AJAX 教程。

ASP - AJAX 与 ASP

AJAX 用于创建动态性更强的应用程序。

AJAX ASP 实例

下面的例子将演示当用户在输入框中键入字符时，网页如何与服务器进行通信：

实例

```
<html>
<head>
<script type="text/javascript">
function showHint(str)
{
var xmlhttp;
if (str.length==0)
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","/ajax/gethint.asp?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>

<h3>请在下面的输入框中键入字母（A - Z）：</h3>
<form action="">
姓氏：<input type="text" id="txt1" onkeyup="showHint(this.value)" />
</form>
<p>建议：<span id="txtHint"></span></p>

</body>
</html>
```

实例解释 - HTML 页面

当用户在上面的输入框中键入字符时，会执行 "showHint()" 函数。该函数由 "onkeyup" 事件触发：

```
<!DOCTYPE html>
<html>
<head>
<script>
function showHint(str)
{
if (str.length==0)
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {// 针对 IE7+, Firefox, Chrome, Opera, Safari 的代码
  xmlhttp=new XMLHttpRequest();
  }
else
  {// 针对 IE6, IE5 的代码
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","gethint.asp?q="+str,true);
xmlhttp.send();
}
</script>
</head
<body>

<p><b>请在输入框中输入英文字符：</b></p>
<form>
First name: <input type="text" onkeyup="showHint(this.value)" size="20">
</form>
<p>Suggestions: <span id="txtHint"></span></p>

</body>
</html>
```

源代码解释：

如果输入框是空的（str.length==0），该函数会清空占位符 txtHint 的内容，并推出该函数。

如果输入框不是空的，那么 showHint() 会执行以下步骤：

- 创建 XMLHttpRequest 对象
- 创建在服务器响应就绪时执行的函数
- 向服务器上的文件发送请求
- 请注意添加到 URL 末端的参数（q）（包含输入框的内容）

ASP 文件

上面这段 JavaScript 调用的服务器页面是名为 "gethint.asp" 的 ASP 文件。

"gethint.asp" 中的源代码会检查姓名数组，然后向浏览器返回对应的姓名：

```
<%
response.expires=-1
dim a(30)
'Fill up array with names
a(1)="Anna"
a(2)="Brittany"
a(3)="Cinderella"
a(4)="Diana"
a(5)="Eva"
a(6)="Fiona"
a(7)="Gunda"
a(8)="Hege"
a(9)="Inga"
a(10)="Johanna"
a(11)="Kitty"
a(12)="Linda"
a(13)="Nina"
a(14)="Ophelia"
a(15)="Petunia"
a(16)="Amanda"
a(17)="Raquel"
a(18)="Cindy"
a(19)="Doris"
a(20)="Eve"
a(21)="Evita"
a(22)="Sunniva"
a(23)="Tove"
a(24)="Unni"
a(25)="Violet"
a(26)="Liza"
a(27)="Elizabeth"
a(28)="Ellen"
a(29)="Wenche"
a(30)="Vicky"

'从 URL 获得参数 q
q=ucase(request.querystring("q"))

'如果长度 q>0，则从数组中查找所有提示
if len(q)>0 then
  hint=""
  for i=1 to 30
    if q=ucase(mid(a(i),1,len(q))) then
      if hint="" then
        hint=a(i)
      else
        hint=hint & " , " & a(i)
      end if
    end if
  next
end if

'如果未找到提示，则输出 "no suggestion"
'or output the correct values
if hint="" then
  response.write("no suggestion")
else
  response.write(hint)
end if
%>
```

源代码解释：

如果 JavaScript 发送了任何文本（即 strlen($q) 大于 0），则会发生：

- 查找匹配来自 JavaScript 的字符的姓名
- 如果未找到匹配，则将响应字符串设置为 "no suggestion"
- 如果找到一个或多个匹配姓名，则用所有姓名设置响应字符串
- 把响应发送到占位符 "txtHint"

AJAX 数据库实例

AJAX 可用来与数据库进行相互的通信。

AJAX 数据库实例

下面的例子演示网页如何通过 AJAX 从数据库中读取信息：

```
<html>
<head>
<script type="text/javascript">
function showCustomer(str)
{
var xmlhttp;
if (str=="")
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
  xmlhttp=new XMLHttpRequest();
  }
else
  {// code for IE6, IE5
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","/ajax/getcustomer.asp?q="+str,true);
xmlhttp.send();
}
</script>
</head>
<body>

<form action="" style="margin-top:15px;">
<label>请选择一位客户：
<select name="customers" onchange="showCustomer(this.value)" style="font-family:Verdana,
<option value="APPLE">Apple Computer, Inc.</option>
<option value="BAIDU ">BAIDU, Inc</option>
<option value="Canon">Canon USA, Inc.</option>
<option value="Google">Google, Inc.</option>
<option value="Nokia">Nokia Corporation</option>
<option value="SONY">Sony Corporation of America</option>
</select>
</label>
</form>
<br />
<div id="txtHint">客户信息将在此处列出 ...</div>

</body>
</html>
```

实例解释 - HTML 页面

当用户在上面的下拉列表中选择某位客户时，会执行名为 "showCustomer()" 的函数。该函数由 "onchange" 事件触发：

```
<!DOCTYPE html>
<html>
<head>
<script>
function showCustomer(str)
{
if (str=="")
  {
  document.getElementById("txtHint").innerHTML="";
  return;
  }
if (window.XMLHttpRequest)
  {// 针对 IE7+, Firefox, Chrome, Opera, Safari 的代码
  xmlhttp=new XMLHttpRequest();
  }
else
  {// 针对 IE6, IE5 的代码
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
xmlhttp.onreadystatechange=function()
  {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
    document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
xmlhttp.open("GET","getcustomer.asp?q="+str,true);
xmlhttp.send();
}
</script>
</head
<body>

<form>
<select name="customers" onchange="showCustomer(this.value)">
<option value="">Select a customer:</option>
<option value="ALFKI">Alfreds Futterkiste</option>
<option value="NORTS ">North/South</option>
<option value="WOLZA">Wolski Zajazd</option>
</select>
</form>
<br>
<div id="txtHint">客户信息将在此处列出...</div>

</body>
</html>
```

源代码解释：

如果没有选择客户（str.length 等于 0），那么该函数会清空 txtHint 占位符，然后退出该函数。

如果已选择一位客户，则 showCustomer() 函数会执行以下步骤：

- 创建 XMLHttpRequest 对象
- 创建在服务器响应就绪时执行的函数

- 向服务器上的文件发送请求
- 请注意添加到 URL 末端的参数（q）（包含下拉列表的内容）

ASP 文件

上面这段 JavaScript 调用的服务器页面是名为 "getcustomer.asp" 的 ASP 文件。

"getcustomer.asp" 中的源代码会运行一次针对数据库的查询，然后在 HTML 表格中返回结果：

```
<%
response.expires=-1
sql="SELECT * FROM CUSTOMERS WHERE CUSTOMERID="
sql=sql & "'" & request.querystring("q") & "'"

set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs=Server.CreateObject("ADODB.recordset")
rs.Open sql,conn

response.write("<table>")
do until rs.EOF
  for each x in rs.Fields
    response.write("<tr><td><b>" & x.name & "</b></td>")
    response.write("<td>" & x.value & "</td></tr>")
  next
  rs.MoveNext
loop
response.write("</table>")
%>
```

ADO 教程

ADO 简介

ADO 被用于从网页访问数据库。

您应当具备的基础知识

在继续学习之前，您需要对下面的知识有基本的了解：

- WWW、HTML 以及对网站构建的基本了解
- ASP（动态服务器页面）
- SQL（结构化查询语言）

如果您希望首先学习这些项目，请在我们的 首页 访问这些教程。

什么是 ADO？

- ADO 是一项微软的技术
- ADO 指 ActiveX 数据对象（_A_ctiveX _D_ata _O_bjects）
- ADO 是一个微软的 Active-X 组件
- ADO 会随微软的 IIS 被自动安装
- ADO 是一个访问数据库中数据的编程接口

从 ASP 页面访问数据库

从一个 ASP 页面内部访问数据库的通常的方法是：

1. 创建一个到数据库的 ADO 连接
2. 打开数据库连接
3. 创建 ADO 记录集
4. 从记录集提取您需要的数据
5. 关闭记录集
6. 关闭连接

ADO 数据库连接

在从某个网页访问数据之前，必须先建立一个数据库连接。

创建一个 DSN-less 数据库连接

连接到某一个数据库的最简单的方法是使用一个 DSN-less 连接。DSN-less 连接可被用于您的站点上的任何微软 Access 数据库。

假设您拥有一个名为 "northwind.mdb" 的数据库位于 "c:/webdata/" 的 web 目录中，您可以使用下面的 ASP 代码连接到此数据库：

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
%>
```

注意，在上面的例子中，您必须规定微软的 Access 数据库驱动程序（Provider），以及此数据库在计算机上的物理路径。

创建一个 ODBC 数据库连接

假设您拥有一个名为 "northwind" 的 ODBC 数据库，您可以使用下面的 ASP 代码连接到此数据库：

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Open "northwind"
%>
```

通过一个 ODBC 连接，您可以连接到您的网络中任何计算机上的任何数据库，只要 ODBC 连接是可用的。

到 MS Access 数据库的 ODBC 连接

下面为您讲解如何创建到一个 MS Access 数据库的连接：

1. 打开控制面板中的 *ODBC* 图标
2. 选择系统 *ODBC* 选项卡
3. 点击 ODBC 选项卡中的添加按钮

4. 选择 Driver to Microsoft Access，然后点击完成按钮
5. 在下一个窗口中点击“选择”按钮来定位数据库
6. 为此数据库赋予一个数据源名称（_D_ata _S_ource _N_ame，DSN）
7. 点击"确定"

注意：此配置必须在您的网站所在的计算机上完成。假如您正在自己的计算机上运行PWS或者IIS，此架构是可以运行的，但是假如您的网站位于一台远程的服务器，您就必须拥有此服务器的物理访问权限，或者请您的 web 主机提供商为您做这些事情。

ADO 连接对象（ADO Connection Object）

ADO 连接对象用来创建到某个数据源的开放连接。通过此连接，您可以对此数据库进行访问和操作。

查看此连接对象的所有方法和属性。

ADO Recordset（记录集）

如需读取数据库的数据，那么其中的数据必须首先被载入一个记录集中。

创建一个 ADO 表记录集（ADO Table Recordset）

在 ADO 数据库连接创建之后，如上一章所述，接下来就可以建立一个 ADO 记录集了。

假设我们有一个名为 "Northwind" 的数据库，我们可以通过下面的代码访问数据库中的 "Customers" 表：

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Customers", conn
%>
```

创建一个 ADO SQL 记录集 （ADO SQL Recordset）

我们也可使用 SQL 访问 "Customers" 表中的数据：

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Select * from Customers", conn
%>
```

从记录集中提取数据

在记录集被打开后，我们可以从记录集中提取数据。

假设我们用一个名为 "Northwind" 的数据库，我们可以通过下面的代码访问数据库中 "Customers" 表：

```
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs=Server.CreateObject("ADODB.recordset")
rs.Open "Select * from Customers", conn

for each x in rs.fields
   response.write(x.name)
   response.write(" = ")
   response.write(x.value)
next
%>
```

ADO 记录集对象（ADO Recordset Object）

ADO Recordset 对象可被用来容纳来自数据库表的记录集。

查看 ADO Recordset 对象的所有方法和属性。

ADO 显示

显示来自记录集中的数据的最常用的方法，就是把数据显示在 **HTML** 表格中。

实例

显示记录

如何首先创建一个数据库连接，然后创建一个记录集，然后把其中的数据显示在HTML中。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
rs.Open "Select * from Customers", conn

do until rs.EOF
    for each x in rs.Fields
       Response.Write(x.name)
       Response.Write(" = ")
       Response.Write(x.value & "<br />")
    next
    Response.Write("<br />")
    rs.MoveNext
loop

rs.close
conn.close
%>

</body>
</html>
```

在 HTML 表格中显示记录

如何把数据表中的数据显示在HTML表格中。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))

set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT Companyname, Contactname FROM Customers", conn
%>

<table border="1" width="100%">
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

向 HTML 表格添加标题

如何向HTML表格添加标题，以使其可读性更强。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers"
rs.Open sql, conn
%>

<table border="1" width="100%">
<tr>
<%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
next%>
</tr>
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

向 HTML 表格添加颜色

如何向HTML表格添加颜色，以使其更加美观。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers"
rs.Open sql, conn
%>

<table border="1" width="100%" bgcolor="#fff5ee">
<tr>
<%for each x in rs.Fields
    response.write("<th align='left' bgcolor='#b0c4de'>" & x.name & "</th>")
next%>
</tr>
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

显示字段名称和字段值

我们有一个名为 "Northwind" 的数据库，并且我们希望显示出 "Customers" 表中的数据（记得以 .asp 为扩展名来保存这个文件）：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT * FROM Customers", conn

do until rs.EOF
  for each x in rs.Fields
    Response.Write(x.name)
    Response.Write(" = ")
    Response.Write(x.value & "<br />")
  next
  Response.Write("<br />")
  rs.MoveNext
loop

rs.close
conn.close
%>

</body>
</html>
```

在一个 HTML 表格中显示字段名称和字段的值

我们也可以通过下面的代码把表 "Customers" 中的数据显示在一个 HTML 表格中：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT Companyname, Contactname FROM Customers", conn
%>

<table border="1" width="100%">
<%do until rs.EOF%>
   <tr>
   <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
   <%next
   rs.MoveNext%>
   </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

向 HTML 表格添加标题

我们希望为这个 HTML 表格添加标题，这样它就更易读了：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers"
rs.Open sql, conn
%>

<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
  next%>
  </tr>
  <%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
  <%loop
  rs.close
  conn.close
  %>
</table>

</body>
</html>
```

ADO 查询

我们可以使用 **SQL** 来创建查询，这样就可以指定仅查看选定的记录和字段。

实例

显示 "Companyname" 以 A 开头的记录

如何仅仅显示 "Customers" 表的 "Companyname" 字段中以 A 开头的记录。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers WHERE CompanyName LIKE 'A%'"
rs.Open sql, conn
%>

<table border="1" width="100%">
<tr>
<%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
next%>
</tr>
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

显示 "Companyname" 大于 E 的记录

如何仅仅显示 "Customers" 表的 "Companyname" 字段中大于 E 的记录。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers WHERE CompanyName>'E'"
rs.Open sql, conn
%>

<table border="1" width="100%">
<tr>
<%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
next%>
</tr>
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%> </td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

仅显示西班牙的客户

如何仅仅显示 "Customers" 表中的西班牙客户。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers WHERE Country='China'"
rs.Open sql, conn
%>

<table border="1" width="100%">
<tr>
<%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
next%>
</tr>
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%> </td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

让用户来选择筛选标准

让用户根据国别来选择客户

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))

set rs=Server.CreateObject("ADODB.recordset")
sql="SELECT DISTINCT Country FROM Customers ORDER BY Country"
rs.Open sql,conn

country=request.form("country")

%>

<form method="post">
Choose Country <select name="country">
<%  do until rs.EOF
    response.write("<option")
    if rs.fields("country")=country then
      response.write(" selected")
    end if
    response.write(">")
    response.write(rs.fields("Country"))
    rs.MoveNext
loop
rs.Close
set rs=Nothing %>
</select>
<input type="submit" value="Show customers">
</form>

<%
if country<>"" then
   sql="SELECT Companyname,Contactname,Country FROM Customers WHERE country='" & country
   set rs=Server.CreateObject("ADODB.Recordset")
   rs.Open sql,conn
%>
   <table width="100%" cellspacing="0" cellpadding="2" border="1">
   <tr>
     <th>Companyname</th>
     <th>Contactname</th>
     <th>Country</th>
   </tr>
<%
do until rs.EOF
   response.write("<tr>")
   response.write("<td>" & rs.fields("companyname") & "</td>")
   response.write("<td>" & rs.fields("contactname") & "</td>")
   response.write("<td>" & rs.fields("country") & "</td>")
   response.write("</tr>")
   rs.MoveNext
loop
rs.close
conn.Close
set rs=Nothing
set conn=Nothing%>
</table>
<%  end if %>

</body>
</html>
```

显示选定的数据

我们希望仅仅显示 "Customers" 表的 "Companyname" 字段中以 A 开头的记录：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs=Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers
WHERE CompanyName LIKE 'A%'"
rs.Open sql, conn
%>

<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
  next%>
  </tr>
  <%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
  <%loop
  rs.close
  conn.close%>
</table>

</body>
</html>
```

ADO 排序

我们可以使用**SQL**来规定如何对记录集中的数据进行排序。

实例

根据指定的字段名处对记录进行升序排序

如何根据指定字段名对数据进行排序

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers ORDER BY CompanyName"
rs.Open sql, conn
%>

<table border="1" width="100%">
<tr>
<%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
next%>
</tr>
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

根据指定的字段名处对记录进行降序排序

如何根据指定字段名对数据进行排序

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM Customers ORDER BY CompanyName DESC"
rs.Open sql, conn
%>

<table border="1" width="100%">
<tr>
<%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
next%>
</tr>
<%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
       <td><%Response.Write(x.value)%> </td>
    <%next
    rs.MoveNext%>
    </tr>
<%loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

让用户来选择根据哪列进行排序

让用户来选择根据哪列进行排序

```
<html>
<body>

<table border="1" width="100%" bgcolor="#fff5ee">
<tr>
<th align="left" bgcolor="#b0c4de">
<a href="/example/adoe/demo_adoe_sort_3.asp?sort=companyname">Company</a>
</th>
<th align="left" bgcolor="#b0c4de">
<a href="/example/adoe/demo_adoe_sort_3.asp?sort=contactname">Contact</a>
</th>
</tr>
<%
if request.querystring("sort")<>"" then
   sort=request.querystring("sort")
else
   sort="companyname"
end if

set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs=Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname,Contactname FROM Customers ORDER BY " & sort
rs.Open sql,conn

do until rs.EOF
   response.write("<tr>")
   for each x in rs.Fields
     response.write("<td>" & x.value & "</td>")
   next
   rs.MoveNext
   response.write("</tr>")
loop
rs.close
conn.close
%>
</table>

</body>
</html>
```

对数据进行排序

我们希望显示 "Customers" 表中的"Companyname"和"Contactname"字段，并根据"Companyname"进行排序（请记得用.asp为后缀保存）：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
sql="SELECT Companyname, Contactname FROM
Customers ORDER BY CompanyName"
rs.Open sql, conn
%>

<table border="1" width="100%">
  <tr>
  <%for each x in rs.Fields
    response.write("<th>" & x.name & "</th>")
  next%>
  </tr>
  <%do until rs.EOF%>
    <tr>
    <%for each x in rs.Fields%>
      <td><%Response.Write(x.value)%></td>
    <%next
    rs.MoveNext%>
    </tr>
  <%loop
  rs.close
  conn.close%>
</table>

</body>
</html>
```

ADO 添加记录

我们可以使用 SQL 的 **INSERT INTO** 命令向数据库中的表添加记录。

向数据库中的表添加记录

我们希望向 Northwind 数据库中的 Customers 表添加一条新的记录。我们首先要创建一个表单，这个表单包含了我们需要从中搜集数据的输入域：

```
<html>
<body>

<form method="post" action="demo_add.asp">
<table>
<tr>
<td>CustomerID:</td>
<td><input name="custid"></td>
</tr><tr>
<td>Company Name:</td>
<td><input name="compname"></td>
</tr><tr>
<td>Contact Name:</td>
<td><input name="contname"></td>
</tr><tr>
<td>Address:</td>
<td><input name="address"></td>
</tr><tr>
<td>City:</td>
<td><input name="city"></td>
</tr><tr>
<td>Postal Code:</td>
<td><input name="postcode"></td>
</tr><tr>
<td>Country:</td>
<td><input name="country"></td>
</tr>
</table>
<br /><br />
<input type="submit" value="Add New">
<input type="reset" value="Cancel">
</form>

</body>
</html>
```

当用户按下确认按钮时，这个表单就会被送往名为 "demo_add.asp" 的文件。文件 "demo_add.asp" 中含有可向 Customers 表添加一条新记录的代码：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

sql="INSERT INTO customers (customerID,companyname,"
sql=sql & "contactname,address,city,postalcode,country)"
sql=sql & " VALUES "
sql=sql & "('" & Request.Form("custid") & "',"
sql=sql & "'" & Request.Form("compname") & "',"
sql=sql & "'" & Request.Form("contname") & "',"
sql=sql & "'" & Request.Form("address") & "',"
sql=sql & "'" & Request.Form("city") & "',"
sql=sql & "'" & Request.Form("postcode") & "',"
sql=sql & "'" & Request.Form("country") & "')"

on error resume next
conn.Execute sql,recaffected
if err<>0 then
  Response.Write("No update permissions!")
else
  Response.Write("<h3>" & recaffected & " record added</h3>")
end if
conn.close
%>

</body>
</html>
```

重要事项

在您使用 INSERT command 命令时，请注意以下事项：

- 如果表含有一个主键，请确保向主键字段添加的值是唯一且非空的（否则，provider 就不会追加此记录，亦或发生错误）
- 如果表含有一个自动编号的字段，请不要在 INSERT 命令中涉及此字段（这个字段的值是由 provider 负责的）

关于无数据字段

在 MS Access 数据库中，假如您将 AllowZeroLength 属性设置为 “Yes”，您可以在文本、超链接以及备忘字段输入零长度的字符串 ("")。

注释：并非所有的数据库都支持零长度的字符串，因而当添加带有空白字段的记录时可能会产生错误。因此，检查您使用的数据库所支持的数据类型是很重要的。

ADO 更新记录

我们可使用 **SQL** 的 **UPDATE** 来更新数据库表中的某条记录。

更新数据库表中的记录

我们希望更新 Northwind 数据中 Customers 表的某条记录。首先我们需要创建一个表格，来列出 Customers 中的所有记录。

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.Recordset")
rs.open "SELECT * FROM customers",conn
%>

<h2>List Database</h2>
<table border="1" width="100%">
<tr>
<%
for each x in rs.Fields
  response.write("<th>" & ucase(x.name) & "</th>")
next
%>
</tr>
<% do until rs.EOF %>
<tr>
<form method="post" action="demo_update.asp">
<%
for each x in rs.Fields
  if lcase(x.name)="customerid" then%>
    <td>
    <input type="submit" name="customerID" value="<%=x.value%>">
    </td>
  <%else%>
    <td><%Response.Write(x.value)%></td>
  <%end if
next
%>
</form>
<%rs.MoveNext%>
</tr>
<%
loop
conn.close
%>
</table>

</body>
</html>
```

如果用户点击 "customerID" 列中的按钮，会打开一个新文件 "demo_update.asp"。此文件包含了创建输入域的源代码，这些输入域基于数据库中记录的字段，同时也含有一个保存修改的“更新按钮”：

```
<html>
<body>

<h2>Update Record</h2>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

cid=Request.Form("customerID")

if Request.form("companyname")="" then
  set rs=Server.CreateObject("ADODB.Recordset")
  rs.open "SELECT * FROM customers WHERE customerID='" & cid & "'",conn
  %>
  <form method="post" action="demo_update.asp">
  <table>
  <%for each x in rs.Fields%>
  <tr>
  <td><%=x.name%></td>
  <td><input name="<%=x.name%>" value="<%=x.value%>"></td>
  <%next%>
  </tr>
  </table>
  <br /><br />
  <input type="submit" value="Update record">
  </form>
<%
else
  sql="UPDATE customers SET "
  sql=sql & "companyname='" & Request.Form("companyname") & "',"
  sql=sql & "contactname='" & Request.Form("contactname") & "',"
  sql=sql & "address='" & Request.Form("address") & "',"
  sql=sql & "city='" & Request.Form("city") & "',"
  sql=sql & "postalcode='" & Request.Form("postalcode") & "',"
  sql=sql & "country='" & Request.Form("country") & "'"
  sql=sql & " WHERE customerID='" & cid & "'"
  on error resume next
  conn.Execute sql
  if err<>0 then
    response.write("No update permissions!")
  else
    response.write("Record " & cid & " was updated!")
  end if
end if
conn.close
%>

</body>
</html>
```

ADO 删除记录

我们可使用 **SQL** 的 **DELETE** 命令来删除数据库表中的某条记录。

删除表中的记录

我们希望删除 Northwind 数据库的 Customers 表中的一条记录。首先我们需要创建一个表格，来列出 Customers 中的所有记录。

```
<html>
<body>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"
set rs=Server.CreateObject("ADODB.Recordset")
rs.open "SELECT * FROM customers",conn
%>

<h2>List Database</h2>
<table border="1" width="100%">
<tr>
<%
for each x in rs.Fields
  response.write("<th>" & ucase(x.name) & "</th>")
next
%>
</tr>
<% do until rs.EOF %>
<tr>
<form method="post" action="demo_delete.asp">
<%
for each x in rs.Fields
  if x.name="customerID" then%>
    <td>
    <input type="submit" name="customerID" value="<%=x.value%>">
    </td>
  <%else%>
    <td><%Response.Write(x.value)%></td>
  <%end if
next
%>
</form>
<%rs.MoveNext%>
</tr>
<%
loop
conn.close
%>
</table>

</body>
</html>
```

假如用户点击 "customerID" 列中的按钮，会打开新文件 "demo_delete.asp"。此文件包含了创建输入域的源代码，这些输入域基于数据库中记录的字段，同时也含有一个删除当前记录的“删除按钮”：

```
<html>
<body>

<h2>Delete Record</h2>
<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

cid=Request.Form("customerID"

if Request.form("companyname")="" then
  set rs=Server.CreateObject("ADODB.Recordset")
  rs.open "SELECT * FROM customers WHERE customerID='" & cid & "'",conn
  %>
  <form method="post" action="demo_delete.asp">
  <table>
  <%for each x in rs.Fields%>
  <tr>
  <td><%=x.name%></td>
  <td><input name="<%=x.name%>" value="<%=x.value%>"></td>
  <%next%>
  </tr>
  </table>
  <br /><br />
  <input type="submit" value="Delete record">
  </form>
<%
else
  sql="DELETE FROM customers"
  sql=sql & " WHERE customerID='" & cid & "'"
  on error resume next
  conn.Execute sql
  if err<>0 then
    response.write("No update permissions!")
  else
    response.write("Record " & cid & " was deleted!")
  end if
end if
conn.close
%>

</body>
</html>
```

ADO 通过 GetString() 加速脚本

请使用 **GetString()** 方法来加速您的 **ASP** 脚本（来代替多行的 **Response.Write**）。

实例

使用 GetString()

如何使用 GetString() 在 HTML 表格中显示记录集中的数据。

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open(Server.Mappath("/db/northwind.mdb"))
set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT Companyname, Contactname FROM Customers", conn
str=rs.GetString(,,"</td><td>","</td></tr><tr><td>"," ")
%>

<table border="1" width="100%">
   <tr>
     <td><%Response.Write(str)%></td>
   </tr>
</table>

<%
rs.close
conn.close
set rs = Nothing
set conn = Nothing
%>

</body>
</html>
```

多行 Response.Write

下面的例子演示了在 HTML 表格中显示数据库查询的一种方法：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT Companyname, Contactname FROM Customers", conn
%>

<table border="1" width="100%">
<%do until rs.EOF%>
   <tr>
      <td><%Response.Write(rs.fields("Companyname"))%></td>
      <td><%Response.Write(rs.fields("Contactname"))%></td>
   </tr>
<%rs.MoveNext
loop%>
</table>

<%
rs.close
conn.close
set rs = Nothing
set conn = Nothing
%>

</body>
</html>
```

对于一个大型的查询来说，这样做会增加脚本的处理时间，这是由于服务器需要处理大量的 Response.Write 命令。

解决的办法是创建全部字符串，从 <table> 到 </table>，然后将其输出 - 只使用一次 Response.Write。

GetString() 方法

GetString() 方法使我们有能力仅使用一次 Response.Write，就可以显示所有的字符串。同时它甚至不需要 do..loop 代码以及条件测试来检查记录集是否处于 EOF。

语法

```
str = rs.GetString(format,rows,coldel,rowdel,nullexpr)
```

如需使用来自记录集的数据创建一个 HTML 表格，我们仅仅需要使用以上参数中的三个（所有的参数都是可选的）：

- coldel - 用作列分隔符的 HTML
- rowdel - 用作行分隔符的 HTML
- nullexpr - 当列为空时所使用的 HTML

注释：GetString() 方法是 ADO 2.0 的特性。您可从下面的地址下载 ADO 2.0：http://www.microsoft.com/data/download.htm

在下面的例子中，我们将使用 GetString() 方法，把记录集存为一个字符串：

```
<html>
<body>

<%
set conn=Server.CreateObject("ADODB.Connection")
conn.Provider="Microsoft.Jet.OLEDB.4.0"
conn.Open "c:/webdata/northwind.mdb"

set rs = Server.CreateObject("ADODB.recordset")
rs.Open "SELECT Companyname, Contactname FROM Customers", conn

str=rs.GetString(,,"</td><td>","</td></tr><tr><td>"," ")
%>

<table border="1" width="100%">
  <tr>
    <td><%Response.Write(str)%></td>
  </tr>
</table>

<%
rs.close
conn.close
set rs = Nothing
set conn = Nothing
%>
</body>
</html>
```

上面的变量 str 包含着由 SELECT 语句返回的所有列和行的一个字符串。在每列之间会出现 </td><td>，在每行之间会出现 </td></tr><tr><td>。这样，仅使用一次 Response.Write，我们就得到了需要的 HTML。

ADO 对象

ADO Command 对象

Command 对象

ADO Command 对象用于执行面向数据库的一次简单查询。此查询可执行诸如创建、添加、取回、删除或更新记录等动作。

如果该查询用于取回数据，此数据将以一个 RecordSet 对象返回。这意味着被取回的数据能够被 RecordSet 对象的属性、集合、方法或事件进行操作。

Command 对象的主要特性是有能力使用存储查询和带有参数的存储过程。

ProgID

```
set objCommand=Server.CreateObject("ADODB.command")
```

属性

属性	描述
ActiveConnection	设置或返回包含了定义连接或 Connection 对象的字符串。
CommandText	设置或返回包含提供者（provider）命令（如 SQL 语句、表格名称或存储的过程调用）的字符串值。默认值为 ""（零长度字符串）。
CommandTimeout	设置或返回长整型值，该值指示等待命令执行的时间（单位为秒）。默认值为 30。
CommandType	设置或返回一个 Command 对象的类型
Name	设置或返回一个 Command 对象的名称
Prepared	指示执行前是否保存命令的编译版本（已经准备好的版本）。
State	返回一个值，此值可描述该 Command 对象处于打开、关闭、连接、执行还是取回数据的状态。

方法

方法	描述
Cancel	取消一个方法的一次执行。
CreateParameter	创建一个新的 Parameter 对象
Execute	执行 CommandText 属性中的查询、SQL 语句或存储过程。

集合

集合	描述
Parameters	包含一个 Command 对象的所有 Parameter 对象。
Properties	包含一个 Command 对象的所有 Property 对象。

ADO Connection 对象

Connection 对象

ADO Connection 对象用于创建一个到达某个数据源的开放连接。通过此连接，您可以对一个数据库进行访问和操作。

如果需要多次访问某个数据库，您应当使用 Connection 对象来建立一个连接。您也可以经由一个 Command 或 Recordset 对象传递一个连接字符串来创建某个连接。不过，此类连接仅仅适合一次具体的简单的查询。

ProgID

```
set objConnection=Server.CreateObject("ADODB.connection")
```

属性

属性	描述
Attributes	设置或返回 Connection 对象的属性。
CommandTimeout	指示在终止尝试和产生错误之前执行命令期间需等待的时间。
ConnectionString	设置或返回用于建立连接数据源的细节信息。
ConnectionTimeout	指示在终止尝试和产生错误前建立连接期间所等待的时间。
CursorLocation	设置或返回游标服务的位置。
DefaultDatabase	指示 Connection 对象的默认数据库。
IsolationLevel	指示 Connection 对象的隔离级别。
Mode	设置或返回 provider 的访问权限。
Provider	设置或返回 Connection 对象提供者的名称。
State	返回一个描述连接是打开还是关闭的值。
Version	返回 ADO 的版本号。

方法

方法	描述
BeginTrans	开始一个新事务。
Cancel	取消一次执行。
Close	关闭一个连接。
CommitTrans	保存任何更改并结束当前事务。
Execute	执行查询、SQL 语句、存储过程或 provider 具体文本。
Open	打开一个连接。
OpenSchema	从 provider 返回有关数据源的 schema 信息。
RollbackTrans	取消当前事务中所作的任何更改并结束事务。

事件

注释：您无法使用 VBScript or JScript 来处理事件（仅能使用 Visual Basic、Visual C++ 以及 Visual J++ 语言处理事件）。

事件	描述
BeginTransComplete	在 BeginTrans 操作之后被触发。
CommitTransComplete	在 CommitTrans 操作之后被触发。
ConnectComplete	在一个连接开始后被触发。
Disconnect	在一个连接结束之后被触发。
ExecuteComplete	在一条命令执行完毕后被触发。
InfoMessage	假如在一个 ConnectionEvent 操作过程中警告发生，则触发该事件。
RollbackTransComplete	在 RollbackTrans 操作之后被触发。
WillConnect	在一个连接开始之前被触发。
WillExecute	在一条命令被执行之前被触发。

集合

集合	描述
Errors	包含 Connection 对象的所有 Error 对象。
Properties	包含 Connection 对象的所有 Property 对象。

ADO Error 对象

Error 对象

ADO Error 对象包含与单个操作（涉及提供者）有关的数据访问错误的详细信息。

ADO 会因每次错误产生一个 Error 对象。每个 Error 对象包含具体错误的详细信息，且 Error 对象被存储在 Errors 集合中。要访问这些错误，就必须引用某个具体的连接。

循环遍历 Errors 集合：

```
<%
for each objErr in objConn.Errors
  response.write("<p>")
  response.write("Description: ")
  response.write(objErr.Description & "<br />")
  response.write("Help context: ")
  response.write(objErr.HelpContext & "<br />")
  response.write("Help file: ")
  response.write(objErr.HelpFile & "<br />")
  response.write("Native error: ")
  response.write(objErr.NativeError & "<br />")
  response.write("Error number: ")
  response.write(objErr.Number & "<br />")
  response.write("Error source: ")
  response.write(objErr.Source & "<br />")
  response.write("SQL state: ")
  response.write(objErr.SQLState & "<br />")
  response.write("</p>")
next
%>
```

语法

```
objErr.property
```

属性

属性	描述
Description	返回一个错误描述。
HelpContext	返回 Microsoft Windows help system 中某个主题的内容 ID。
HelpFile	返回 Microsoft Windows help system 中帮助文件的完整路径。
NativeError	返回来自 provider 或数据源的错误代码。
Number	返回可标识错误的一个唯一的数字。
Source	返回产生错误的对象或应用程序的名称。
SQLState	返回一个 5 字符的 SQL 错误码。

ADO Field 对象

Field 对象

ADO Field 对象包含有关 Recordset 对象中某一列的信息。Recordset 中的每一列对应一个 Field 对象。

ProgID

```
set objField=Server.CreateObject("ADODB.field")
```

属性

属性	描述
ActualSize	返回一个字段值的实际长度。
Attributes	设置或返回 Field 对象的属性。
DefinedSize	返回Field 对象被定义的大小
Name	设置或返回 Field 对象的名称。
NumericScale	设置或返回 Field 对象中的值所允许的小数位数。
OriginalValue	返回某个字段的原始值。
Precision	设置或返回当表示 Field 对象中的数值时所允许的数字的最大数。
Status	返回 Field 对象的状态。
Type	设置或返回 Field 对象的类型。
UnderlyingValue	返回一个字段的当前值。
Value	设置或返回 Field 对象的值。

方法

方法	描述
AppendChunk	把大型的二进制或文本数据追加到 Field 对象
GetChunk	返回大型二进制或文本 Field 对象的全部或部分内容。

集合

集合	描述
Properties	包含一个 Field 对象的所有 Property 对象。

ADO Parameter 对象

Parameter 对象

ADO Parameter 对象可提供有关被用于存储过程或查询中的一个单个参数的信息。

Parameter 对象在其被创建时被添加到 Parameters 集合。Parameters 集合与一个具体的 Command 对象相关联，Command 对象使用此集合在存储过程和查询内外传递参数。

参数被用来创建参数化的命令。这些命令（在它们已被定义和存储之后）使用参数在命令执行前来改变命令的某些细节。例如，SQL SELECT 语句可使用参数定义 WHERE 子句的匹配条件，而使用另一个参数来定义 SORT BY 子句的列的名称。

有四种类型的参数：input 参数、output 参数、input/output 参数 以及 return 参数。

语法

```
objectname.property
objectname.method
```

属性

属性	描述
Attributes	设置或返回一个 Parameter 对象的属性。
Direction	设置或返回某个参数如何传递到存储过程或从存储过程传递回来。
Name	设置或返回一个 Parameter 对象的名称。
NumericScale	设置或返回一个 Parameter 对象的数值的小数点右侧的数字数目。
Precision	设置或返回当表示一个参数中数值时所允许数字的最大数目。
Size	设置或返回 Parameter 对象中的值的最大大小（按字节或字符）。
Type	设置或返回一个 Parameter 对象的类型。
Value	设置或返回一个 Parameter 对象的值。

方法

方法	描述
AppendChunk	把长二进制或字符数据追加到一个 Parameter 对象。
Delete	从 Parameters 集合中删除一个对象。

ADO Property 对象

Property 对象

ADO 对象有两种类型的属性：内置属性和动态属性。

内置属性是在 ADO 中实现并立即可用于任何新对象的属性，此时使用 MyObject.Property 语法。它们不会作为 Property 对象出现在对象的 Properties 集合中，因此，虽然可以更改它们的值，但无法更改它们的特性。

ADO Property 对象表示 ADO 对象的动态特性，这种动态特性是被 provider 定义的。

每个与 ADO 对话的 provider 拥有不同的方式与 ADO 进行交互。所以，ADO 需要通过某种方式来存储有关 provider 的信息。解决方法是 provider 为 ADO 提供具体的信息（动态属性）。ADO 把每个 provider 属性存储在一个 Property 对象中，而 Property 对象相应地也被存储在 Properties 集合中。此集合会被分配到 Command 对象、Connection 对象、Field 对象 或者 Recordset 对象。

例如，指定给提供者的属性可能会指示 Recordset 对象是否支持事务或更新。这些附加的属性将作为 Property 对象出现在该 Recordset 对象的 Properties 集合中。

ProgID

```
set objProperty=Server.CreateObject("ADODB.property")
```

属性

属性	描述
Attributes	返回一个 Property 对象的属性
Name	设置或返回一个 Property 对象的名称
Type	返回 Property 的类型
Value	设置或返回 一个 Property 对象的值

ADO Record 对象

Record 对象 (ADO version 2.5)

ADO Record 对象用于容纳记录集中的一行、或文件系统的一个文件或一个目录。

ADO 2.5 之前的版本仅能够访问结构化的数据库。在一个结构化的数据库中，每个表在每一行均有确切相同的列数，并且每一列都由相同的数据类型组成。

Record 对象允许访问行与行之间的列数且/或数据类型不同的数据集。

语法

```
objectname.property
objectname.method
```

属性

属性	描述
ActiveConnection	设置或返回 Record 对象当前所属的 Connection 对象。
Mode	设置或返回在Record 对象中修改数据的有效权限。
ParentURL	返回父 Record 的绝对URL。
RecordType	返回 Record 对象的类型。
Source	设置或返回 Record 对象的 Open 方法的 src 参数。
State	返回 Record 对象的状态。

方法

方法	描述
Cancel	取消一次 CopyRecord、DeleteRecord、MoveRecord 或 Open 调用的执行。
Close	关闭一个 Record 对象。
CopyRecord	把文件或目录拷贝到另外一个位置。
DeleteRecord	删除一个文件或目录。
GetChildren	返回一个 Recordset 对象，其中的每一行表示目录中的文件或子目录。
MoveRecord	把文件或目录移动到另外一个位置。
Open	打开一个已有的 Record 对象或创建一个新的文件或目录。

集合

集合	描述
Properties	特定提供者属性的一个集合。
Fields	包含 Record 对象中的所有 Field 对象。

Fields 集合的属性

属性	描述
Count	返回 fields 集合中的项目数。起始值为 0。例子： `countfields = rec.Fields.Count`
Item(named_item/number)	返回 fields 集合中的某个指定的项目。例子： `itemfields = rec.Fields.Item(1)` 或者 `itemfields = rec.Fields.Item("Name")`

ADO Recordset 对象

实例

GetRows

本例演示如何使用 GetRows 方法。

Recordset 对象

ADO Recordset 对象用于容纳一个来自数据库表的记录集。一个 Recordset 对象由记录和列（字段）组成。

在 ADO 中，此对象是最重要且最常用于对数据库的数据进行操作的对象。

ProgID

```
set objRecordset=Server.CreateObject("ADODB.recordset")
```

当您首次打开一个 Recordset 时，当前记录指针将指向第一个记录，同时 BOF 和 EOF 属性为 False。如果没有记录，BOF 和 EOF 属性为 True。

Recordset 对象能够支持两种更新类型：

在 ADO，定义了 4 中不同的游标（指针）类型：

- 动态游标 - 允许您查看其他用户所作的添加、更改和删除
- 键集游标 - 类似动态游标，不同的是您无法查看有其他用户所做的添加，并且它会防止您访问其他用户已删除的记录。其他用户所做的数据更改仍然是可见的。
- 静态游标 - 提供记录集的静态副本，可用来查找数据或生成报告。此外，由其他用户所做的添加、更改和删除将是不可见的。当您打开一个客户端 Recordset 对象时，这是唯一被允许的游标类型。
- 仅向前游标 - 只允许在 Recordset 中向前滚动。此外，由其他用户所做的添加、更改和删除将是不可见的。

可通过 CursorType 属性或 Open 方法中的 CursorType 参数来设置游标的类型。

注释：并非所有的提供者（providers）支持 Recordset 对象的所有方法和属性。

属性

属性	描述
AbsolutePage	设置或返回一个可指定 Recordset 对象中页码的值。
AbsolutePosition	设置或返回一个值，此值可指定 Recordset 对象中当前记录的顺序位置（序号位置）。
ActiveCommand	返回与 Recordset 对象相关联的 Command 对象。
ActiveConnection	如果连接被关闭，设置或返回连接的定义，如果连接打开，设置或返回当前的 Connection 对象。
BOF	如果当前的记录位置在第一条记录之前，则返回 true，否则返回 fasle。
Bookmark	设置或返回一个书签。此书签保存当前记录的位置。
CacheSize	设置或返回能够被缓存的记录的数目。
CursorLocation	设置或返回游标服务的位置。
CursorType	设置或返回一个 Recordset 对象的游标类型。
DataMember	设置或返回要从 DataSource 属性所引用的对象中检索的数据成员的名称。
DataSource	指定一个包含要被表示为 Recordset 对象的数据的对象。
EditMode	返回当前记录的编辑状态。
EOF	如果当前记录的位置在最后的记录之后，则返回 true，否则返回 fasle。
Filter	返回一个针对 Recordset 对象中数据的过滤器。
Index	设置或返回 Recordset 对象的当前索引的名称。
LockType	设置或返回当编辑 Recordset 中的一条记录时，可指定锁定类型的值。
MarshalOptions	设置或返回一个值，此值指定哪些记录被返回服务器。
MaxRecords	设置或返回从一个查询返回 Recordset 对象的的最大记录数目。
PageCount	返回一个 Recordset 对象中的数据页数。
PageSize	设置或返回 Recordset 对象的一个单一页面上所允许的最大记录数。
RecordCount	返回一个 Recordset 对象中的记录数目。
Sort	设置或返回一个或多个作为 Recordset 排序基准的字段名。
Source	设置一个字符串值，或一个 Command 对象引用，或返回一个字符串值，此值可指示 Recordset 对象的数据源。
State	返回一个值，此值可描述是否 Recordset 对象是打开、关闭、正在连接、正在执行或正在取回数据。
Status	返回有关批更新或其他大量操作的当前记录的状态。

StayInSync	设置或返回当父记录位置改变时对子记录的引用是否改变。

方法

方法	描述
AddNew	创建一条新记录。
Cancel	撤销一次执行。
CancelBatch	撤销一次批更新。
CancelUpdate	撤销对 Recordset 对象的一条记录所做的更改。
Clone	创建一个已有 Recordset 的副本。
Close	关闭一个 Recordset。
CompareBookmarks	比较两个书签。
Delete	删除一条记录或一组记录。
Find	搜索一个 Recordset 中满足指定某个条件的一条记录。
GetRows	把多条记录从一个 Recordset 对象中拷贝到一个二维数组中。
GetString	将 Recordset 作为字符串返回。
Move	在 Recordset 对象中移动记录指针。
MoveFirst	把记录指针移动到第一条记录。
MoveLast	把记录指针移动到最后一条记录。
MoveNext	把记录指针移动到下一条记录。
MovePrevious	把记录指针移动到上一条记录。
NextRecordset	通过执行一系列命令清除当前 Recordset 对象并返回下一个 Recordset。
Open	打开一个数据库元素，此元素可提供对表的记录、查询的结果或保存的 Recordset 的访问。
Requery	通过重新执行对象所基于的查询来更新 Recordset 对象中的数据。
Resync	从原始数据库刷新当前 Recordset 中的数据。
Save	把 Recordset 对象保存到 file 或 Stream 对象中。
Seek	搜索 Recordset 的索引以快速定位与指定的值相匹配的行，并使其成为当前行。
Supports	返回一个布尔值，此值可定义 Recordset 对象是否支持特定类型的功能。
Update	保存所有对 Recordset 对象中的一条单一记录所做的更改。
UpdateBatch	把所有 Recordset 中的更改存入数据库。请在批更新模式中使用。

事件

Note: You cannot handle events using VBScript or JScript (only Visual Basic, Visual C++, and Visual J++ languages can handle events).

事件	描述
EndOfRecordset	当试图移动到超过 Recordset 结尾的行时被触发。
FetchComplete	当异步操作中的所有记录均被读取后被触发。
FetchProgress	在异步操作期间被定期地触发，报告已读取多少记录。
FieldChangeComplete	Field 对象的值更改被触发。
MoveComplete	Recordset 中的当前位置更改后被触发。
RecordChangeComplete	一条记录更改之后被触发。
RecordsetChangeComplete	在 Recordset 更改之后被触发。
WillChangeField	在 Field 对象的值更改之前被触发
WillChangeRecord	在一条记录更改之前被触发。
WillChangeRecordset	在 Recordset 更改之前被触发。
WillMove	在 Recordset 中的当前位置更改之前被触发。

集合

集合	描述
Fields	指示在此 Recordset 对象中 Field 对象的数目。
Properties	包含所有 Recordset 对象中的 Property 对象。

Fields 集合的属性

属性	描述
Count	返回 fields 集合中项目的数目。以 0 起始。例子： `countfields = rs.Fields.Count`
Item(named_item/number)	返回 fields 集合中的某个指定的项目。例子： `itemfields = rs.Fields.Item(1)` 或者 `itemfields = rs.Fields.Item("Name")`

Properties 集合的属性

属性	描述
Count	返回 properties 集合中项目的数目。以 0 起始。例子：`countprop = rs.Properties.Count`
Item(named_item/number)	返回 properties 集合中某个指定的项目。例子：`itemprop = rs.Properties.Item(1)` 或者 `itemprop = rs.Properties.Item("Name")`

ADO Stream 对象

Stream 对象 (ADO version 2.5)

ADO Stream 对象用于读写以及处理二进制数据或文本流。

Stream 对象可通过三种方法获得：

- 通过指向包含二进制或文本数据的对象（通常是文件）的 URL。此对象可以是简单的文档、表示结构化文档的 Record 对象或文件夹。
- 通过将 Stream 对象实例化。这些 Stream 对象可用来存储用于应用程序的数据。跟与 URL 相关联的 Stream 或 Record 的默认 Stream 不同，实例化的 Stream 在默认情况下与基本源没有关联。
- 通过打开与 Record 对象相关联的默认 Stream 对象。打开 Record 时便可获取与 Record 对象相关联的默认流。只需打开该流便可删除一个往返过程。

语法

```
objectname.property
objectname.method
```

属性

属性	描述
CharSet	指定用于存储 Stream 的字符集。
EOS	返回当前位置是否位于流的结尾。
LineSeparator	设置或返回用在文本 Stream 对象中的分行符。
Mode	设置或返回供修改数据的可用权限。
Position	设置或返回从 Stream 对象开始处的当前位置（按字节计算）。
Size	返回一个打开的 Stream 对象的大小。
State	返回一个描述 Stream 是打开还是关闭的值。
Type	设置或返回 Stream 对象中的数据的类型。

方法

方法	描述
Cancel	取消对 Stream 对象的 Open 调用的执行。
Close	关闭一个 Stream 对象。
CopyTo	把指定数目的字符/比特从一个 Stream 对象拷贝到另外一个 Stream 对象。
Flush	把 Stream 缓冲区中的内容发送到相关联的下层对象。
LoadFromFile	把文件的内容载入 Stream 对象。
Open	打开一个 Stream 对象。
Read	从一个二进制 Stream 对象读取全部流或指定的字节数。
ReadText	从一个文本 Stream 对象中读取全部流、一行或指定的字节数。
SaveToFile	把一个 Stream 对象的二进制内容保存到某个文件。
SetEOS	设置当前位置为流的结尾 (EOS)
SkipLine	在读取一个文本流时跳过一行。
Write	把二进制数据写到一个二进制 Stream 对象。
WriteText	把字符数据写到一个文本 Stream 对象。

ADO 数据类型

下面的表格列出了 Access、SQL Server 与 Oracle 之间的数据类型映射：

DataType Enum	Value	Access	SQLServer	Oracle
adBigInt	20	BigInt (SQL Server 2000 +)		
adBinary	128	Binary TimeStamp	Raw *	
adBoolean	11	YesNo	Bit	
adChar	129	Char	Char	
adCurrency	6	Currency	Money SmallMoney	
adDate	7	Date	DateTime	
adDBTimeStamp	135	DateTime (Access 97 (ODBC))	DateTime SmallDateTime	Date
adDecimal	14	Decimal *		
adDouble	5	Double	Float	Float
adGUID	72	ReplicationID (Access 97 (OLEDB)), (Access 2000 (OLEDB))	UniqueIdentifier (SQL Server 7.0 +)	
adIDispatch	9			
adInteger	3	AutoNumber Integer Long	Identity (SQL Server 6.5) Int	Int *
adLongVarBinary	205	OLEObject	Image	Long Raw * Blob (Oracle 8.1.x)
adLongVarChar	201	Memo (Access 97) Hyperlink (Access 97)	Text	Long * Clob (Oracle 8.1.x)
adLongVarWChar	203	Memo (Access 2000 (OLEDB)) Hyperlink (Access 2000 (OLEDB))	NText (SQL Server 7.0 +)	NClob (Oracle 8.1.x)
		Decimal (Access	Decimal	Decimal Integer

				SmallInt
adSingle	4	Single	Real	
adSmallInt	2	Integer	SmallInt	
adUnsignedTinyInt	17	Byte	TinyInt	
adVarBinary	204	ReplicationID (Access 97)	VarBinary	
adVarChar	200	Text (Access 97)	VarChar	VarChar
adVariant	12	Sql_Variant (SQL Server 2000 +)	VarChar2	
adVarWChar	202	Text (Access 2000 (OLEDB))	NVarChar (SQL Server 7.0 +)	NVarChar2
adWChar	130	NChar (SQL Server 7.0 +)		

- 在 Oracle 8.0.x 中 - decimal 和 int 等于 number 和 number(10)。

ASP 快速参考

来自 **W3School** 的 **ASP** 快速参考。打印出来，放入口袋，以备随时使用。

基础语法

ASP 脚本由 <% 和 %> 包围。这样向浏览器输出内容：

```
<html>
<body>
<% response.write("Hello World!") %>
</body>
</html>
```

ASP 中的默认语言是 VBScript。如需使用其他脚本语言，请在 ASP 页面顶端插入一段语言声明：

```
<%@ language="javascript" %>
<html>
<body>

<%
....
%>
```

表单和用户输入

Request.QueryString 用户收集 method="get" 的表单中的值。从表单通过 GET 发送的信息对所有人都可见（将显示在浏览器的地址栏中），对所发送的数据量也有限制。

Request.Form 用于收集 method="post" 的表单中的值。从表单通过 POST 发送的信息对其他人是不可见，对所发送的数据量没有限制。

ASP Cookies

cookie 常用语识别用户。cookie 是服务器嵌到用户计算机上的小文件。每当相同的计算机通过浏览器请求摸个页面时，也会发送 cookie。

Response.Cookies 命令用于创建 cookie：

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires="May 10,2012"
%>
```

注释：Response.Cookies 命令必须位于 <html> 标签之前！

"Request.Cookies" 命令用于取回 cookie 值：

```
<%
fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)
%>
```

引用文件

通过 #include 指令，在服务器执行前，您能够把一个 ASP 文件的内容插入另一个 ASP 文件中。#include 指令用于创建函数、页头、页脚，或多个页面上重复使用的元素。

语法：

```
<!--#include virtual="somefile.inc"-->
```

或者

```
<!--#include file ="somefile.inc"-->
```

请使用关键词 virtual 来指示以虚拟目录开始的路径。如果名为 "header.inc" 的文件位于名为 /html 的虚拟目录中，那么下面的代码会插入 "header.inc" 的内容：

```
<!-- #include virtual ="/html/header.inc" -->
```

请使用关键词 file 来指示相对路径。相对路径以包含该引用文件的目录开头。如果您的文件位于 html 目录中，而文件 "header.inc" 位于 html\headers 中，下面的代码将在您的文件中插入 "header.inc" 的内容：

```
<!-- #include file ="headers\header.inc" -->
```

请使用关键词 file 与语法 (..) 来引用更高层级目录中的文件。

Global.asa

Global.asa 文件是可选文件，可包含能够由 ASP 应用程序中的每个页面访问的对象声明、变量以及方法。

注释：Global.asa 文件必须存放在 ASP 应用程序的根目录中，而且每个应用程序只能有一个 Global.asa 文件。

Global.asa 文件只能包含以下内容：

- Application 事件
- Session 事件
- <object> 声明
- TypeLibrary 声明

- # include 指令

Application 和 Session 事件

在 Global.asa 中，您可以告诉 application 和 session 对象当 application/session 开始时做什么，当 application/session 结束时做什么。完成该任务的代码位于事件处理程序中。

注释：在 Global.asa 文件中插入代码时，我们并不使用 <% 和 %>，我们需要在 HTML <script> 标签内部放置子程序：

```
<script language="vbscript" runat="server">
sub Application_OnStart
  ' some code
end sub
sub Application_OnEnd
  ' some code
end sub
sub Session_OnStart
  ' some code
end sub
sub Session_OnEnd
  ' some code
end sub
</script>
```

<object> 声明

通过使用 <object> 标签，也可以在 Global.asa 中创建带有 session 或 application 作用域的对象。

注释：<object> 标签应该位于 <script> 标签之外！

语法：

```
<object runat="server" scope="scope" id="id"
{progid="progID"|classid="classID"}>
.......
</object>
```

TypeLibrary 声明

TypeLibrary 是与 COM 对象对应的 DLL 文件的内容容器。通过在 Global.asa 文件中包含对 TypeLibrary 的调用，就能够访问 COM 对象的常量，同时 ASP 代码也能够更好地报告错误。如果您的 Web 应用程序依赖已在类型库中声明了数据类型的 COM 对象，您可以在 Global.asa 中声明该类型库。

语法：

```
<!--
METADATA TYPE="TypeLib"
file="filename"
uuid="typelibraryuuid"
version="versionnumber"
lcid="localeid"
-->
```

Session 对象

Session 对象用于存储有关用户 session 的信息，或者更改其设置。Session 对象中存储的变量存有关于单个用户的信息，并且能够由一个应用程序中的所有页面进行访问。

集合

- Contents - 包含所有通过脚本命令追加到 session 的条目
- StaticObjects - 包含了所有使用 HTML 的 <object> 标签追加到 session 的对象
- Contents.Remove(*item/index*) - 从 Contents 集合删除一个项目
- Contents.RemoveAll() - 从 Contents 集合删除全部项目

属性

- CodePage - 规定显示动态内容时使用的字符集
- LCID - 设置用于显示动态内容的区域标识符
- SessionID - 返回 session id
- Timeout - 设置或返回 session 的超时时间

方法

- Abandon - 撤销 session 对象中的所有对象。

Application 对象

在一起工作以完成某项任务的一组 ASP 文件被称为一个应用程序。ASP 中的 Application 对象用于将这些文件捆绑在一起。所有用户捆绑一个 Application 对象。Application 对象应该存有被应用程序中的许多页面使用的信息（例如数据库连接信息）。

集合

- Contents - 包含所有通过脚本命令追加到应用程序中的项目
- StaticObjects - 包含所有使用 HTML 的 <object> 标签追加到应用程序中的对象
- Contents.Remove - 从 Contents 集合中删除一个项目
- Contents.RemoveAll - 从 Contents 集合中删除所有的项目

方法

- Lock - 防止用户修改 Application 对象中的变量
- Unlock - 允许用户修改 Application 对象中的变量

Response 对象

Response 对象用于从服务器将输出发送给用户。

集合

Cookies(name) - 设置 cookie 的值。假如不存在，就创建 cookie ，然后设置指定的值。

属性

- Buffer - 规定是否缓冲输出。当输出设置缓存时，服务器会阻止向浏览器的响应，直到所有的服务器脚本均被处理，或者直到脚本调用了 Flush 或 End 方法。如果要设置此属性，它应当位于 .asp 文件中的 <html> 标签之前。
- CacheControl - 设置代理服务器是否可以缓存由 ASP 产生的输出。如果设置为 Public，则代理服务器会缓存页面。
- Charset(*charset_name*) - 将字符集的名称追加到 Response 对象中的 content-type 报头。
- ContentType - 设置 Response 对象的 HTTP 内容类型。（比如 "text/html", "image/gif", "image/jpeg", "text/plain"）。默认是 "text/html"
- Expires - 设置页面在失效前的浏览器缓存时间（分钟）

- ExpiresAbsolute - 设置浏览器上页面缓存失效的日期和时间
- IsClientConnected - 指示客户端是否已从服务器断开
- Pics(*pics_label*) - 向 response 报头的 PICS 标志追加值
- Status - 规定由服务器返回的状态行的值

方法

- AddHeader(*name*, *value*) - 向 HTTP 响应添加新的 HTTP 报头和值
- AppendToLog string - 向服务器记录项目（server log entry）的末端添加字符串
- BinaryWrite(*data_to_write*) - 在没有任何字符转换的情况下直接向输出写数据
- Clear - 清除已缓冲的输出。使用该方法来处理错误。如果 Response.Buffer 未设置为 true，该方法将产生 run-time 错误
- End - 停止处理脚本，并返回当前的结果
- Flush - 立即发送已缓存的输出。如果 Response.Buffer 未设置为 true，该方法将产生 run-time 错误
- Redirect(*url*) - 把用户重定向到另一个 URL
- Write(*data_to_write*) - 向用户写文本

Request 对象

当浏览器从服务器请求页面时，就被称为 request。request 对象用于获取来自用户的信息。

集合

- ClientCertificate - 包含了在客户证书中存储的字段值
- Cookies(*name*) - 包含 cookie 值
- Form(*element_name*) - 包含表单值。该表单必须使用 post 方法
- QueryString(*variable_name*) - 包含查询字符串中的变量值
- ServerVariables(*server_variable*) - 包含服务器变量值

属性

- TotalBytes - 返回在请求正文中客户端所发送的字节总数

方法

- BinaryRead - 取回作为 post 请求的一部分而从客户端送往服务器的数据

Server 对象

Server 对象用于访问服务器上的属性和方法。

属性

ScriptTimeout - 设置或返回一段脚本在终止前所能运行多长时间。

方法

- CreateObject(*type_of_object*) - 创建对象的实例
- Execute(*path*) - 从 ASP 文件内部执行另一个 ASP 文件。在被调用的 ASP 文件执行完毕后，控制权返回原先的 ASP 文件
- GetLastError() - 返回描述所发生错误的 ASPError 对象
- HTMLEncode(*string*) - 对字符串应用 HTML 编码
- MapPath(*path*) - 把相对或虚拟路径映射为物理路径
- Transfer(*path*) - 把所有状态信息发送到另一个文件，以备处理。在传送之后，程序的控制权不会返回原先的 ASP 文件
- URLEncode(*string*) - 对字符串应用 URL 编码规则

来源：http://www.w3school.com.cn/asp/asp_quickref.asp

W3School ASP.net 教程

来源：ASP.net教程

整理：飞龙

ASP.NET

经典 ASP - Active Server Pages（动态服务器页面）

ASP， 全称 Active Server Pages（动态服务器页面）， 也被称为经典 ASP， 是在1998年作为微软的第一个服务器端脚本引擎推出的。

ASP 是一种使得网页中的脚本在因特网服务器上被执行的技术。

ASP 页面的文件扩展名是 .asp， 通常是用 VBScript 编写的。

如果您想学习经典 ASP， 请访问我们的 经典 ASP 教程。

ASP.NET

ASP.NET 是新一代 ASP 。它与经典 ASP 是不兼容的，但 ASP.NET 可能包括经典 ASP。

ASP.NET 页面是经过编译的，这使得它们的运行速度比经典 ASP 快。

ASP.NET 具有更好的语言支持，有一大套的用户控件和基于 XML 的组件，并集成了用户身份验证。

ASP.NET 页面的扩展名是 .aspx， 通常是用 VB (Visual Basic) 或者 C# (C sharp) 编写。

在 ASP.NET 中的控件可以用不同的语言（包括 C++ 和 Java）编写。

当浏览器请求 ASP.NET 文件时，ASP.NET 引擎读取文件，编译和执行脚本文件，并将结果以普通的 HTML 页面返回给浏览器。

ASP.NET Razor

Razor 是一种将服务器代码嵌入到 ASP.NET 网页中的新的、简单的标记语法，很像经典 ASP 。

Razor 具有传统的 ASP.NET 的功能，但更容易使用并且更容易学习。

ASP.NET 编程语言

本教程介绍了以下编程语言：

- Visual Basic (VB.NET)
- C# (发音：C sharp)

ASP.NET 服务器技术

本教程介绍了以下服务器技术

- Web Pages（Razor 语法)
- MVC（模型-视图-控制器）
- Web Forms（传统的 ASP.NET）

ASP.NET 开发工具

ASP.NET 支持以下开发工具：

- WebMatrix
- Visual Web Developer
- Visual Studio

在本教程中，Web Pages 教程使用了 WebMatrix， MVC 教程和 Web Forms 教程使用了 Visual Web Developer。

ASP.NET 文件扩展名

- 经典 ASP 文件的文件扩展名为 .asp
- ASP.NET 文件的文件扩展名为 .aspx
- Razor C# 语法的 ASP.NET 文件的文件扩展名为 .cshtml
- Razor VB 语法的 ASP.NET 文件的文件扩展名为 .vbhtml

Web Pages 教程

ASP.NET Web Pages - 教程

ASP.NET 是一个使用 HTML、CSS、JavaScript 和服务器脚本创建网页和网站的开发框架。

ASP.NET 支持三种不同的开发模式：
Web Pages（Web 页面）、MVC（Model View Controller 模型-视图-控制器）、Web Forms（Web 窗体）：

本教程介绍 **Web Pages**。

Web Pages
MVC
Web Forms

从何入手？

多数开发人员学习一个新技术，是从查看运行实例开始的。

通过"运行实例"轻松学习

我们的"运行实例"工具让 Web Pages 变得更简单易学。

它在运行实例的同时显示 ASP.NET 代码和 HTML 输出。

点击"运行实例"按钮来看看它是如何工作的：

Web Pages 实例

```
<html>
<body>
<h1>Hello Web Pages</h1>
<p>The time is @DateTime.Now</p>
</body>
</html>
```

运行实例？

什么是 Web Pages？

Web Pages 是三种创建 ASP.NET 网站和 Web 应用程序的编程模式中的一种。

其他两种编程模式是 Web Forms 和 MVC（Model View Controller 模型-视图-控制器）。

Web Pages 是开发 ASP.NET 网页最简单的开发模式。它提供了一种简单的方式来将 HTML、CSS、JavaScript 和服务器脚本结合起来：

- 容易学习，容易理解，容易使用
- 围绕着单一的网页创建
- 与 PHP 和经典 ASP 相似
- Visual Basic 或者 C# 的服务器脚本
- 全 HTML、CSS 和 JavaScript 控制

Web Pages 内置了数据库、视频、图形、社交媒体和其他更多的 Web Helpers，因此很容易扩展。

Web Pages 教程

如果您刚接触 ASP.NET，建议从 Web Pages 开始学习。

在我们的 Web Pages 教程中，您将学习到如何使用 VB(Visual Basic) 或者 C#(C sharp) 最新的 Razor 服务器标记语法将 HTML、CSS、JavaScript 和服务器代码结合起来。

您也可以学习如何使用具有可编程的 Web Helpers（包括数据库、视频、图形、社交媒体等等）来扩展您的网页。

Web Pages 实例

通过实例学习！

由于 ASP.NET 代码是在服务器上执行的，您不能在您的浏览器中查看代码。您只能看到普通的 HTML 页面输出。

在 w3cschool.cc 中，每个实例都会把隐藏的 ASP.NET 代码显示出来，这将让您更容易地理解它是如何工作的。

Web Pages 实例

Web Pages 参考手册

在本教程的最后，您将看到一套完整的 ASP.NET 参考手册，介绍了对象、组件、属性和方法。

Web Pages 参考手册

使用 WebMatrix

在本教程中，我们使用了 WebMatrix 。

WebMatrix 是一个简单但功能强大的，由微软专门为 Web Pages 量身定做的，免费的 ASP.NET 开发工具。

WebMatrix 包含：

- Web Pages 实例和模板
- 一种 Web 服务器语言（VB 或者 C# 的 Razor 服务器标记语法）
- 一种 Web 服务器（IIS Express）
- 一种数据库服务器（SQL Server Compact）
- 一个完整的 Web 开发框架（ASP.NET）

通过使用 WebMatrix ，您可以从一个空的网站和一个空白页面开始开发，或者您也可以使用"Web 应用程序库"中的开源应用程序进行二次开发。PHP 和 ASP.NET 应用程序很多都是开源的，比如 Umbraco、DotNetNuke、Drupal、Joomla、WordPress 等等。WebMatrix 也有内置安全性、搜索引擎优化和网络出版工具。

使用 WebMatrix 开发的技术和代码可以无缝地转化为完全专业化的 ASP.NET 应用程序。

如果您想尝试使用 WebMatrix ，请点击下面的链接进行安装：

http://www.microsoft.com/web/gallery/install.aspx?appid=WebMatrix

ASP.NET Web Pages - 添加 Razor 代码

在本教程中，我们将使用 C# 和 Visual Basic 代码的 Razor 标记。

什么是 Razor ？

- Razor 是一种将基于服务器的代码添加到网页中的标记语法
- Razor 具有传统 ASP.NET 标记的功能，但更容易使用并且更容易学习
- Razor 是一种服务器端标记语法，与 ASP 和 PHP 很像
- Razor 支持 C# 和 Visual Basic 编程语言

添加 Razor 代码

请记住上一章实例中的网页：

```
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8" />
<title>Web Pages Demo</title>
</head>
<body>
<h1>Hello Web Pages</h1>
</body>
</html>
```

现在向实例中添加一些 Razor 代码：

实例

```
<!DOCTYPE html>

<html lang="en">
<head>
<meta charset="utf-8" />
<title>Web Pages Demo</title>
</head>
<body>
<h1>Hello Web Pages</h1>
<p>The time is @DateTime.Now</p>
</body>
</html>
```

运行实例？

该页面中包含普通的 HTML 标记，除此之外，还添加了一个 @ 标识的 Razor 代码。

Razor 代码能够在服务器上实时地完成多有的动作，并将结果显示出来。（您可以指定格式化选项，否则只会显示默认项。）

主要的 Razor C# 语法规则

- Razor 代码块包含在 @{ ... } 中
- 内联表达式（变量和函数）以 @ 开头
- 代码语句用分号结束
- 变量使用 var 关键字声明
- 字符串用引号括起来
- C# 代码区分大小写
- C# 文件的扩展名是 .cshtml

C# 实例

```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

运行实例 ?

主要的 Razor VB 语法规则

- Razor 代码块包含在 @Code ... End Code 中
- 内联表达式（变量和函数）以 @ 开头
- 变量使用 Dim 关键字声明
- 字符串用引号括起来
- VB 代码不区分大小写
- VB 文件的扩展名是 .vbhtml

实例

```
<!-- Single statement block -->
@Code dim myMessage = "Hello World" End Code

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@Code
dim greeting = "Welcome to our site!"
dim weekDay = DateTime.Now.DayOfWeek
dim greetingMessage = greeting & " Today is: " & weekDay
End Code

<p>The greeting is: @greetingMessage</p>
```

运行实例？

更多关于 C# 和 Visual Basic

如果您想学习更多关于 Razor、C#、Visual Basic 编程语言，请查看本教程的 Razor 部分。

ASP.NET Web Pages - 页面布局

通过 Web Pages， 创建一个布局一致的网站是很容易的事。

一致的外观

在因特网上，您会发现很过网站都具有一致的外观和风格：

- 每个页面有相同的头部
- 每个页面有相同的底部
- 每个页面有相同的样式和布局

通过 Web Pages， 您能非常高效地做到这点。您可以把重复使用的内容块（比如页面头部和底部）写在一个单独的文件中。

您还可以使用布局模板（布局文件）为站点的所有网页定义一致的布局。

Content Blocks（内容块）

许多网站都有一些内容是被显示在站点的每个页面中（比如页面头部和底部）。

通过 Web Pages，您可以使用 **@RenderPage()** 方法从不同的文件导入内容。

内容块（来自另一个文件）能被导入网页中的任何地方。内容块可以包含文本，标记和代码，就像任何普通的网页一样。

将共同的头部和底部写成单独的文件，这样会帮您节省大量的工作。您不必在每个页面中书写相同的内容，当内容有变动时，您只要修改头部或者底部文件，就可以看到站点中的每个页面的相应内容都已更新。

以下显示了它在代码中是如何呈现的：

实例

```
<html>
<body>
@RenderPage("header.cshtml")
<h1>Hello Web Pages</h1>
<p>This is a paragraph</p>
@RenderPage("footer.cshtml")
</body>
</html>
```

运行实例？

Layout Page（布局页）

在上一部分，您看到了，想在多个网页中显示相同内容是非常容易的。

另一种创建一致外观的方法是使用布局页。一个布局页包含了网页的结构，而不是内容。当一个网页（内容页）链接到布局页，它会根据布局页（模板）的结构进行显示。

布局页中使用 **@RenderBody()** 方法嵌入内容页，除此之外，它与一个正常的网页没有什么差别。

每个内容页都必须以布局指令开始。

以下显示了它在代码中是如何呈现的：

布局页：

```
<html>
<body>
<p>This is header text</p>
@RenderBody()
<p>&copy; 2012 W3CSchool. All rights reserved.</p>
</body>
</html>
```

任何网页：

```
@{Layout="Layout.cshtml";}

<h1>Welcome to W3CSchool.cc</h1>

<p>
Lorem ipsum dolor sit amet, consectetur adipisicing elit,sed do eiusmod tempor incididunt
</p>
```

运行实例？

D.R.Y. - Don't Repeat Yourself（不要自我重复）

通过 Content Blocks（内容块）和 Layout Pages（布局页）这两个 ASP.NET 工具，您可以让您的 Web 应用程序显示一致的外观。

这两个工具能帮您节省大量的工作，您不必再每个页面上重复相同的信息。集中的标记、样式和代码让您的 Web 应用程序更易于管理，更易于维护。

防止文件被浏览

在 ASP.NET 中，文件的名称以下划线开头，可以防止这些文件在网上被浏览。

如果您不想让您的内容块或者布局页被您的用户看到，可以重命名这些文件：

_header.cshtm

_footer.cshtml

_Layout.cshtml

隐藏敏感信息

在 ASP.NET 中，隐藏敏感信息（数据库密码、电子邮件密码等等）最通用的方法是将这些信息保存在一个名为"_AppStart"的单独的文件中。

_AppStart.cshtml

```
@{
WebMail.SmtpServer = "mailserver.example.com";
WebMail.EnableSsl = true;
WebMail.UserName = "username@example.com";
WebMail.Password = "your-password";
WebMail.From = "your-name-here@example.com";
}
```

ASP.NET Web Pages - 文件夹

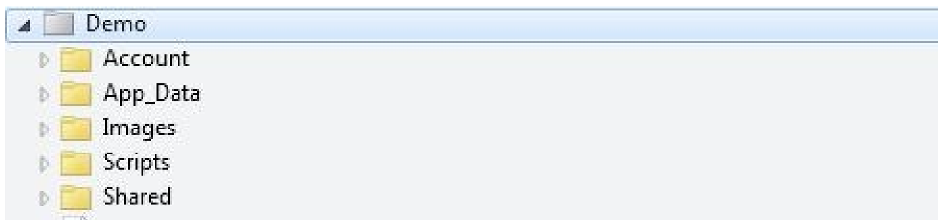
本章介绍有关文件夹和文件夹路径的知识。

在本章中，您将学到：

- 逻辑文件夹结构和物理文件夹结构
- 虚拟名称和物理名称
- Web URL 和 Web 路径

逻辑文件夹结构

下面是典型的 ASP.NET 网站文件夹结构：



- "Account" 文件夹包含登录和安全文件
- "App_Data" 文件夹包含数据库和数据文件
- "Images" 文件夹包含图片
- "Scripts" 文件夹包含浏览器脚本
- "Shared" 文件夹包含公共的文件（比如布局和样式文件）

物理文件夹结构

在上述网站中的"Images"文件夹在计算机上的物理文件夹结构可能如下：

C:\Documents\MyWebSites\Demo\Images

虚拟名称和物理名称

以上面的例子为例：

网站图片的虚拟名称可能是"../img/pic31.jpg"。

对应的物理名称是"C:\Documents\MyWebSites\Demo\Images\pic31.jpg"。

URL 和路径

URL 是用来访问网站中的文件： http://www.w3cschool.cc/html/html-tutorial.html

URL 对应于服务器上的物理文件：C:\MyWebSites\w3cschool\html\html-tutorial.html

虚拟路径是物理路径的一种简写表示。如果您使用虚拟路径，当您更改域名或者将您的网页移到其他服务器上时，您可以不用更新路径。

URL	http://www.w3cschool.cc/html/html-tutorial.html
服务器名称	w3cschool
虚拟路径	/html/html-tutorial.html
物理路径	C:\MyWebSites\w3cschool\html\html-tutorial.html

磁盘驱动器的根目录如下书写 C:，但是网站的根目录是 / （斜线）。

Web 文件夹的虚拟路径通常是与物理文件夹不相同。

在您的代码中，根据您的编码需要决定使用物理路径和和虚拟路径。

ASP.NET 文件夹路径有 3 种工具：~ 运算符、Server.MapPath 方法 和 Href 方法。

~ 运算符

使用 ~ 运算符，在编程代码中规定虚拟路径。

如果您使用 ~ 运算符，在您的站点迁移到其他不同的文件夹或者位置时，您可以不用更改您的任何代码：

```
var myImagesFolder = "~/images";
var myStyleSheet = "~/styles/StyleSheet.css";
```

Server.MapPath 方法

Server.MapPath 方法将虚拟路径（/index.html）转换成服务器能理解的物理路径（C:\Documents\MyWebSites\Demo\default.html）。

当您需要打开服务器上的数据文件时，您可以使用这个方法（只有提供完整的物理路径才能访问数据文件）：

```
var pathName = "~/dataFile.txt";
var fileName = Server.MapPath(pathName);
```

在本教程的下一章中，您会学到更多关于读取（和写入）服务器上的数据文件的知识。

Href 方法

Href 方法将代码中的使用的路径转换成浏览器可以理解的路径（浏览器无法理解 ~ 运算符）。

您可以使用 Href 方法创建资源（比如图像文件 和 CSS 文件）的路径。

一般会在 HTML 中的 <a>、<img> 和 <link> 元素中使用此方法:

```
@{var myStyleSheet = "~/Shared/Site.css";}

<!-- This creates a link to the CSS file. -->
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />

<!-- Same as : -->
<link rel="stylesheet" type="text/css" href="/Shared/Site.css" />
```

Href 方法是 WebPage 对象的一种方法。

ASP.NET Web Pages - 全局页面

本章介绍全局页面 AppStart 和 PageStart。

在 Web 启动之前：_AppStart

大多数的服务器端代码是写在个人网页里边。例如，如果网页中包含输入表单，那么这个网页通常包含用来读取表单数据的服务器端代码。

然而，您可以通过在您的站点根目录下创建一个名为 _AppStart 的页面，这样在站点启动之前可以先启动代码执行。如果存在此页面，ASP.NET 会在站点中其它页面被请求时，优先运行这个页面。

_AppStart 的典型用途是启动代码和初始化全局数值（比如计数器和全局名称）。

注释 **1**：_AppStart 的文件扩展名与您的网页一致，比如：_AppStart.cshtml。

注释 **2**：_AppStart 有下划线前缀。因此，这些文件不可以直接浏览。

在每一个页面之前：_PageStart

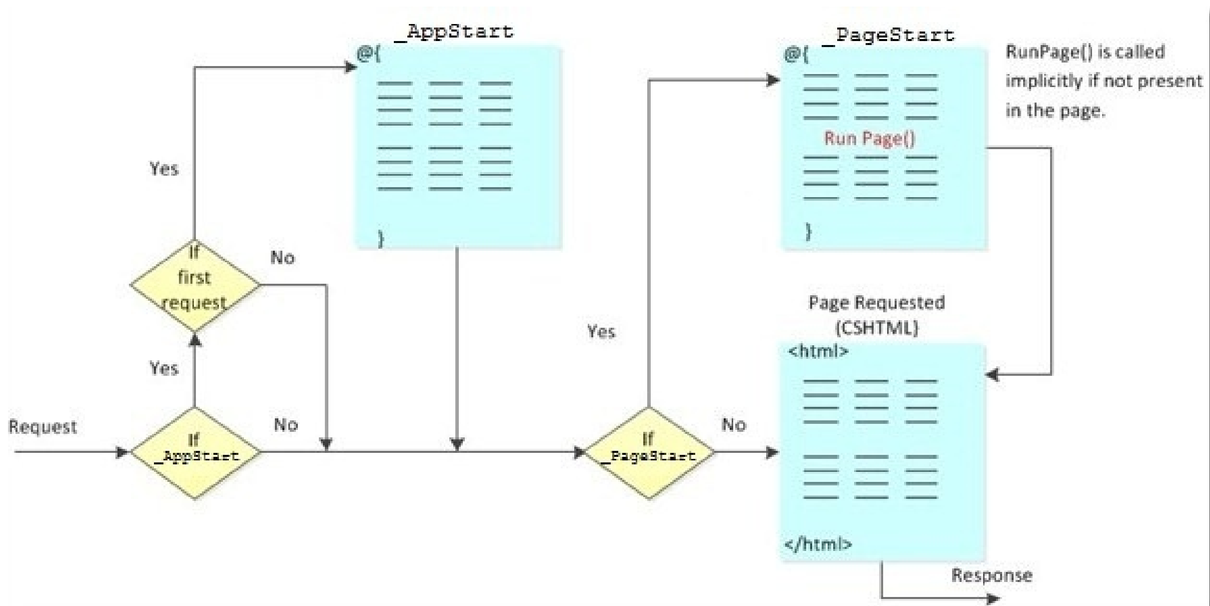
就像 _AppStart 在您的站点启动之前就运行一样，您可以编写在每个文件夹中的任何页面之前运行的代码。

对于您网站中的每个文件夹，您可以添加一个名为 _PageStart 的文件。

_PageStart 的典型用途是为一个文件夹中的所有页面设置布局页面，或者在运行某个页面之前检查用户是否已经登录。

它是如何工作的？

下图显示了它是如何工作的：



当接收到一个请求时，ASP.NET 会首先检查 _AppStart 是否存在。 如果 _AppStart 存在且这是站点接收到的第一个请求，则运行 _AppStart。

然后 ASP.NET 检查 _PageStart 是否存在。如果 _PageStart 存在，则在其它被请求的页面运行之前先运行 _PageStart。

您可以在 _PageStart 中调用 RunPage() 来指定被请求页面的运行位置。否则，默认情况下，被请求页面是在 _PageStart 运行之后才被运行。

ASP.NET Web Pages - HTML 表单

表单是 HTML 文档中放置输入控件（文本框、复选框、单选按钮、下拉列表）的部分。

创建一个 HTML 输入页面

Razor 实例

```
<html>
<body>
@{
if (IsPost) {
string companyname = Request["companyname"];
string contactname = Request["contactname"];
<p>You entered: <br />
Company Name: @companyname <br />
Contact Name: @contactname </p>
}
else
{
<form method="post" action="">
Company Name:<br />
<input type="text" name="CompanyName" value="" /><br />
Contact Name:<br />
<input type="text" name="ContactName" value="" /><br /><br />
<input type="submit" value="Submit" class="submit" />
</form>
}
}
</body>
</html>
```

运行实例？

Razor 实例 - 显示图像

假设在您的图像文件夹中有 3 张图像，您想根据用户的选择动态地显示图像。

这可以通过一段简单的 Razor 代码来实现。

如果在您的网站的图像文件夹中有一个名为 "Photo1.jpg" 的图像，您可以使用 HTML 的 <img> 元素来显示图像，如下所示：



下面的例子演示了如何显示用户从下列列表中选择的图像：

Razor 实例

```
@{
var imagePath="";
if (Request["Choice"] != null)
{imagePath="../img/" + Request["Choice"];}
}
<!DOCTYPE html>
<html>
<body>
<h1>Display Images</h1>
<form method="post" action="">
I want to see:
<select name="Choice">
<option value="Photo1.jpg">Photo 1</option>
<option value="Photo2.jpg">Photo 2</option>
<option value="Photo3.jpg">Photo 3</option>
</select>
<input type="submit" value="Submit" />
@if (imagePath != "")
{
<p>
<img src="@imagePath" alt="Sample" />
</p>
}
</form>
</body>
</html>
```

运行实例？

实例解释

服务器创建了一个叫 **imagePath** 的变量。

HTML 页面有一个名为 **Choice** 的下拉列表（<select> 元素）。它允许用户根据自己的意愿选择一个名称（如 **Photo 1**），当页面被提交到 Web 服务器时，则传递了一个文件名（如 **Photo1.jpg**）。

Razor 代码通过 **Request["Choice"]** 读取 Choice 的值。如果通过代码构建的图像路径（../img/Photo1.jpg）有效，就把图像路径赋值给变量 **imagePath**。

在 HTML 页面中，<img> 元素用来显示图像。当页面显示时，src 属性用来设置 imagePath 变量的值。

<img> 元素是在一个 if 块中，这是为了防止显示没有名称的图像，比如页面第一次被加载显示的时候。

ASP.NET Web Pages - 对象

Web Pages 经常是跟对象有关的。

Page 对象

您已经看到了一些在使用的 Page 对象方法：

```
@RenderPage("header.cshtml")

@RenderBody()
```

在前面的章节中，您已经看到了两个 Page 对象属性（isPost 和 Request）：

```
If (isPost) {

if (Request["Choice"] != null {
```

某些 Page 对象方法

方法	描述
href	使用指定的值创建 URL。
RenderBody()	呈现不在布局页命名区域的内容页的一部分。
RenderPage(*page*)	在另一个页面中呈现某一个页面的内容。
RenderSection(*section*)	呈现布局页命名区域的内容。
Write(*object*)	将对象作为 HTML 编码字符串写入。
WriteLiteral	写入对象时优先不使用 HTML 编码。

某些 Page 对象属性

属性	描述
isPost	如果客户端使用的 HTTP 数据传输方法是 POST 请求，则返回 true。
Layout	获取或者设置布局页面的路径。
Page	提供了对页面和布局页之间共享的数据的类似属性访问。
Request	为当前的 HTTP 请求获取 HttpRequest 对象。
Server	获取 HttpServerUtility 对象，该对象提供了网页处理方法。

Page 对象的 Page 属性

Page 对象的 Page 属性，提供了对页面和布局页之间共享的数据的类似属性访问。

您可以对 Page 属性使用（添加）您自己的属性：

- Page.Title
- Page.Version
- Page.anythingyoulike

页面属性是非常有用的。例如，在内容文件中设置页面标题，并在布局文件中使用：

Home.cshtml

```
@{
Layout="~/Shared/Layout.cshtml";
Page.Title="Home Page"
}

<h1>Welcome to W3CSchool.cc</h1>

<h2>Web Site Main Ingredients</h2>

<p>A Home Page (Default.cshtml)</p>
<p>A Layout File (Layout.cshtml)</p>
<p>A Style Sheet (Site.css)</p>
```

Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
<title>@Page.Title</title>
</head>
<body>
@RenderBody()
</body>
</html
```

ASP.NET Web Pages - 文件

本章介绍有关使用文本文件的知识。

使用文本文件

在前面的章节中，我们已经了解到网页数据是存储在数据库中的。

您也可以把站点数据存储在文本文件中。

用来存储数据的文本文件通常被称为平面文件。常见的文本文件格式是 .txt、.xml 和 .csv（逗号分隔值）。

在本章中，您将学习到：

- 如何从文本文件中读取并显示数据

手动添加一个文本文件

在下面的例子中，您将需要一个文本文件。

在您的网站上，如果没有 App_Data 文件夹，请创建一个。在 App_Data 文件夹中，创建一个名为 Persons.txt 的文件。

添加以下内容到文件中：

Persons.txt

```
George,Lucas
Steven,Spielberg
Alfred,Hitchcock
```

显示文本文件中的数据

下面的实例演示了如何显示一个文本文件中的数据：

实例

```
@{
var dataFile = Server.MapPath("~/App_Data/Persons.txt");
Array userData = File.ReadAllLines(dataFile);
}

<!DOCTYPE html>
<html>
<body>

<h1>Reading Data from a File</h1>
@foreach (string dataLine in userData)
{
foreach (string dataItem in dataLine.Split(','))
{@dataItem <text> </text>}
<br />
}
</body>
</html>
```

运行实例？

实例解释

使用 **Server.MapPath** 找到确切的文本文件的路径。

使用 **File.ReadAllLines** 打开文本文件，并读取文件中的所有行到一个数组中。

数组中的每个数据行中的数据项的数据被显示。

显示 Excel 文件中的数据

使用 Microsoft Excel，您可以将一个电子表格保存为一个逗号分隔的文本文件（.csv 文件）。此时，电子表格中的每一行保存为一个文本行，每个数据列由逗号分隔。

in可以使用上面的实例读取一个 Excel .csv 文件（只需将文件名改成相应的 Excel 文件的名称）。

ASP.NET Web Pages - 帮助器

Web 帮助器大大简化了 Web 开发和常见的编程任务。

ASP.NET 帮助器

ASP.NET 帮助器是通过几行简单的 Razor 代码即可访问的组件。

您可以使用存放在 .cshtml 文件中的 Razor 语法构建自己的帮助器，或者使用内建的 ASP.NET 帮助器。

在本教程接下来的章节中，您将学到如何使用 Razor 帮助器。

下面是一些有用的 Razor 帮助器的简短说明：

WebGrid 帮助器

WebGrid 帮助器简化了显示数据的方式：

- 自动创建一个 HTML 表格来显示数据
- 支持不同的格式化选项
- 支持数据分页显示（第一页、下一页、上一页、最后一页）
- 支持通过点击列表标题进行排序

Chart 帮助器

"Chart 帮助器" 能显示不同类型的带有多种格式化选项和标签的图表图像。

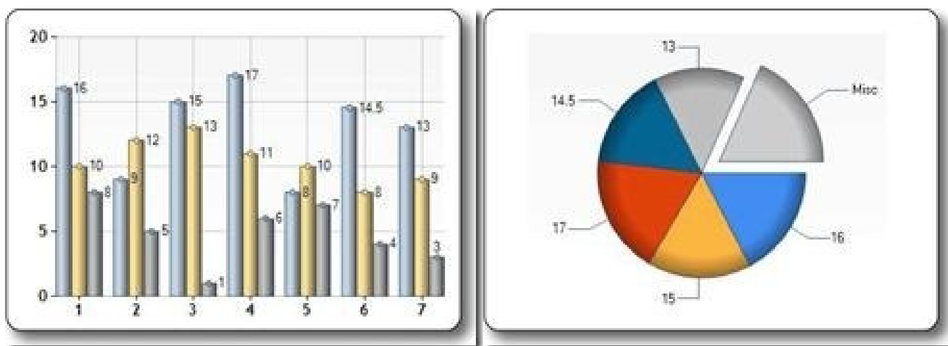


Chart 帮助器显示的数据来源可以是数组、数据库或者文件。

WebMail 帮助器

WebMail 帮助器提供了使用SMTP（Simple Mail Transfer Protocol 简单邮件传输协议）发送电子邮件的功能。

WebImage 帮助器

WebImage 帮助器提供了管理网页中图像的功能。

关键词：翻转、旋转、缩放、水印。

第三方帮助器

通过 Razor，您可以利用内建的或者第三方的帮助器来简化电子邮件、数据库、多媒体、社交网络以及很多其他问题（如导航和的网络安全）的使用。

安装帮助器

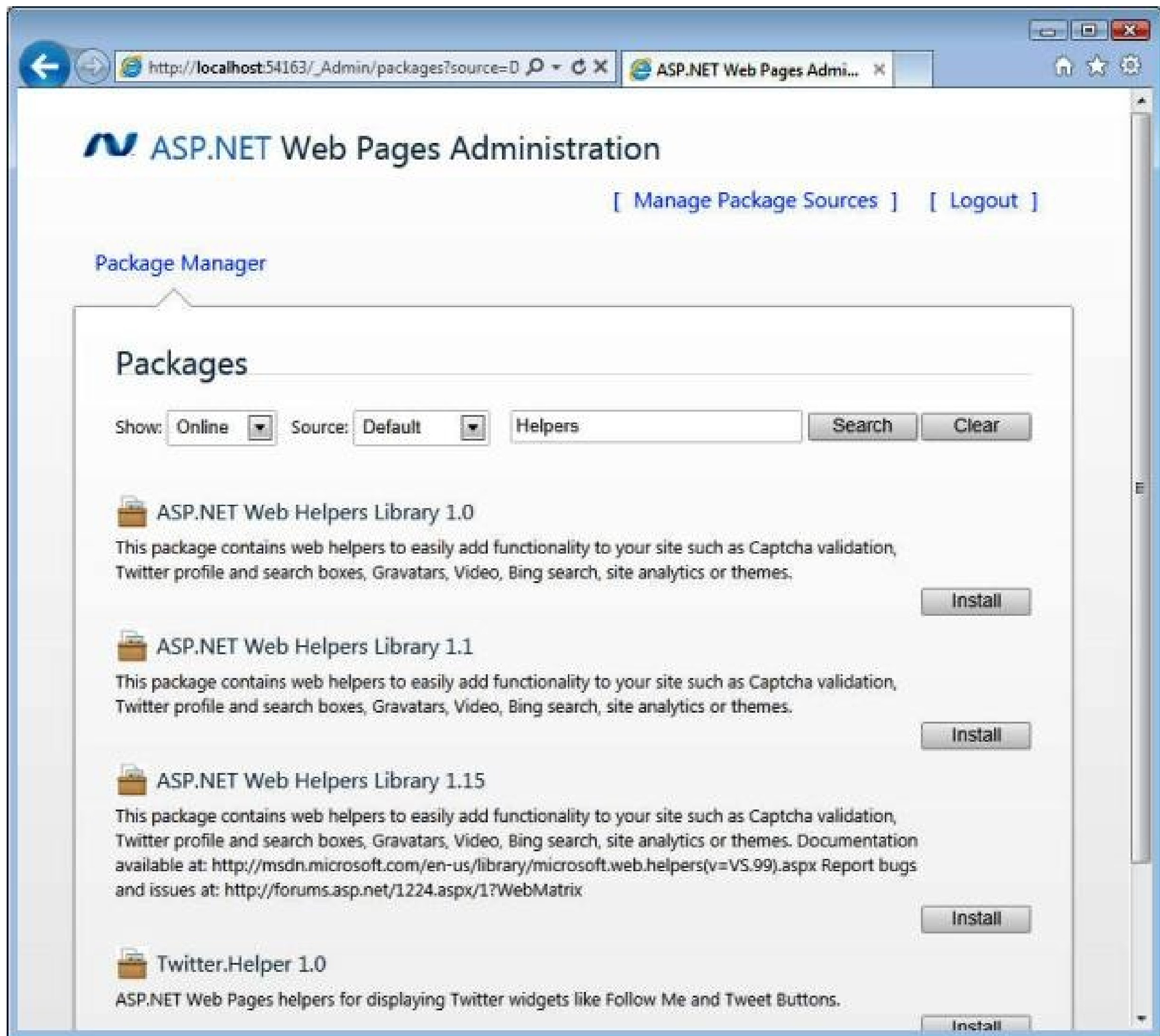
WebMatrix 已经包含了一些帮助器，您还可以手动安装其他的帮助器。

在 w3cschool.cc 的 WebPages 帮助器参考手册中，您可以看到一个便捷的参考手册，包含了内建帮助器和其他可以通过手动安装附加到 ASP.NET Web Helpers Library 工具包中的帮助器。

如果您在 WebMatrix 中创建了一个网站，请按照下面的步骤安装帮助器：

1. 在 WebMatrix 中，打开 **Site** 工作区。
2. 点击 **Web Pages Administration**。
3. 使用密码 * 登录到 Web Pages Administration。
4. 使用 搜索区 搜索帮助器。
5. 点击 **Install** 安装您所需的帮助器。

（* 如果您是第一次使用 Web Pages Administration，会提示您创建一个密码。）


http://localhost:54163/_Admin/packages?source=D
ASP.NET Web Pages Admi...
ASP.NET Web Pages Administration
[Manage Package Sources]
[Logout]
Package Manager
Packages
Show: Online
Source: Default
Helpers
Search
Clear
ASP.NET Web Helpers Library 1.0
This package contains web helpers to easily add functionality to your site such as Captcha validation, Twitter profile and search boxes, Gravatars, Video, Bing search, site analytics or themes.
Install
ASP.NET Web Helpers Library 1.1
This package contains web helpers to easily add functionality to your site such as Captcha validation, Twitter profile and search boxes, Gravatars, Video, Bing search, site analytics or themes.
Install
ASP.NET Web Helpers Library 1.15
This package contains web helpers to easily add functionality to your site such as Captcha validation, Twitter profile and search boxes, Gravatars, Video, Bing search, site analytics or themes. Documentation available at: http://msdn.microsoft.com/en-us/library/microsoft.web.helpers(v=VS.99).aspx Report bugs and issues at: http://forums.asp.net/1224.aspx/1?WebMatrix
Install
Twitter.Helper 1.0
ASP.NET Web Pages helpers for displaying Twitter widgets like Follow Me and Tweet Buttons.

ASP.NET Web Pages - WebGrid 帮助器

WebGrid - 众多有用的 ASP.NET Web 帮助器之一。

自己写的 HTML

在前面的章节中，您使用 Razor 代码显示数据库数据，所有的 HTML 标记都是手写的：

数据库实例

```
@{
var db = Database.Open("SmallBakery");
var selectQueryString = "SELECT * FROM Product ORDER BY Name";
}
<html>
<body>
<h1>Small Bakery Products</h1>
<table>
<tr>
<th>Id</th>
<th>Product</th>
<th>Description</th>
<th>Price</th>
</tr>
@foreach(var row in db.Query(selectQueryString))
{
<tr>
<td>@row.Id</td>
<td>@row.Name</td>
<td>@row.Description</td>
<td align="right">@row.Price</td>
</tr>
}
</table>
</body>
</html>
```

运行实例？

使用 WebGrid 帮助器

WebGrid 帮助器提供了一种更简单的显示数据的方法。

WebGrid 帮助器：

- 自动创建一个 HTML 表格来显示数据
- 支持不同的格式化选项
- 支持数据分页显示
- 支持通过点击列表标题进行排序

WebGrid 实例

```
@{
var db = Database.Open("SmallBakery") ;
var selectQueryString = "SELECT * FROM Product ORDER BY Id";
var data = db.Query(selectQueryString);
var grid = new WebGrid(data);
}
<html>
<head>
<title>Displaying Data Using the WebGrid Helper</title>
</head>
<body>
<h1>Small Bakery Products</h1>
<div id="grid">
@grid.GetHtml()
</div>
</body>
</html>
```

运行实例？

ASP.NET Web Pages - Chart 帮助器

Chart 帮助器 - 众多有用的 ASP.NET Web 帮助器之一。

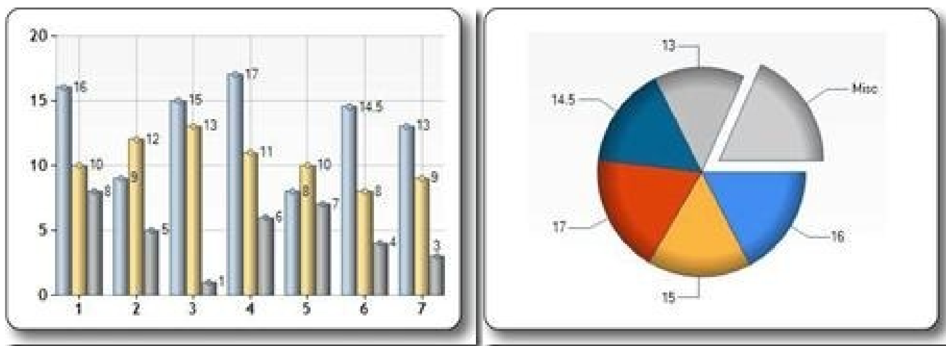
Chart 帮助器

在前面的章节中，您已经学习了如何使用 ASP.NET 的 "帮助器"。

前面已经介绍了如何使用 "WebGrid 帮助器" 在网格中显示数据。

本章介绍如何使用 "Chart 帮助器" 以图形化的形式显示数据。

"Chart 帮助器" 可以创建不同类型的带有多种格式化选项和标签的图表图像。它可以创建面积图、条形图、柱形图、折线图、饼图等标准图表，也可以创建像股票图表这样的更专业的图表。



在图表中显示的数据可以是来自一个数组，一个数据库，或者一个文件中的数据。

根据数组创建图表

下面的实例显示了根据数组数据显示图表所需的代码：

实例

```
@{
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Employees")
.AddSeries(chartType: "column",
xValue: new[] { "Peter", "Andrew", "Julie", "Mary", "Dave" },
yValues: new[] { "2", "6", "4", "5", "3" })
.Write();
}
```

运行实例？

- **new Chart** 创建一个新的图表对象并且设置它的宽度和高度
- **AddTitle** 方法指定了图表的标题
- **AddSeries** 方法向图表中增加数据
- **chartType** 参数定义图表的类型
- **xValue** 参数定义 x 轴的名称
- **yValues** 参数定义 y 轴的名称
- **Write()** 方法显示图表

根据数据库创建图表

您可以执行一个数据库查询，然后使用查询结果中的数据来创建一个图表：

实例

```
@{
var db = Database.Open("SmallBakery");
var dbdata = db.Query("SELECT Name, Price FROM Product");
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Product Sales")
.DataBindTable(dataSource: dbdata, xField: "Name")
.Write();
}
```

运行实例？

- **var db = Database.Open** 打开数据库（将数据库对象赋值给变量 db）
- **var dbdata = db.Query** 执行数据库查询并保存结果在 dbdata 中
- **new Chart** 创建一个新的图表对象并且设置它的宽度和高度
- **AddTitle** 方法指定了图表的标题
- **DataBindTable** 方法将数据源绑定到图表
- **Write()** 方法显示图表

除了使用 DataBindTable 方法之外，另一种方法是使用 AddSeries（见前面的实例）。DataBindTable 更容易使用，但是 AddSeries 更加灵活，因为您可以更明确地指定图表和数据：

实例

```
@{
var db = Database.Open("SmallBakery");
var dbdata = db.Query("SELECT Name, Price FROM Product");
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Product Sales")
.AddSeries(chartType:"Pie",
xValue: dbdata, xField: "Name",
yValues: dbdata, yFields: "Price")
.Write();
}
```

运行实例？

根据 XML 数据创建图表

第三种创建图表的方法是使用 XML 文件作为图表的数据：

实例

```
@using System.Data;

@{
var dataSet = new DataSet();
dataSet.ReadXmlSchema(Server.MapPath("data.xsd"));
dataSet.ReadXml(Server.MapPath("data.xml"));
var dataView = new DataView(dataSet.Tables[0]);
var myChart = new Chart(width: 600, height: 400)
.AddTitle("Sales Per Employee")
.AddSeries("Default", chartType: "Pie",
xValue: dataView, xField: "Name",
yValues: dataView, yFields: "Sales")
.Write();}
}
```

运行实例？

ASP.NET Web Pages - WebMail 帮助器

WebMail 帮助器 - 众多有用的 ASP.NET Web 帮助器之一。

WebMail 帮助器

WebMail 帮助器让发送邮件变得更简单，它按照 SMTP（Simple Mail Transfer Protocol 简单邮件传输协议）从 Web 应用程序发送邮件。

前提：电子邮件支持

为了演示如何使用电子邮件，我们将创建一个输入页面，让用户提交一个页面到另一个页面，并发送一封关于支持问题的邮件。

第一：编辑您的 AppStart 页面

如果在本教程中您已经创建了 Demo 应用程序，那么您已经有一个名为 _AppStart.cshtml 的页面，内容如下：

_AppStart.cshtml

```
@{
WebSecurity.InitializeDatabaseConnection("Users", "UserProfile", "UserId", "Email", true)
}
```

要启动 WebMail 帮助器，向您的 AppStart 页面中增加如下所示的 WebMail 属性：

_AppStart.cshtml

```
@{
WebSecurity.InitializeDatabaseConnection("Users", "UserProfile", "UserId", "Email", true)
WebMail.SmtpServer = "smtp.example.com";
WebMail.SmtpPort = 25;
WebMail.EnableSsl = false;
WebMail.UserName = "support@example.com";
WebMail.Password = "password-goes-here";
WebMail.From = "john@example.com";
}
```

属性解释：

SmtpServer: 用于发送电子邮件的 SMTP 服务器的名称。

SmtpPort: 服务器用来发送 SMTP 事务（电子邮件）的端口。

EnableSsl: 如果服务器使用 SSL（Secure Socket Layer 安全套接层）加密，则值为 true。

UserName: 用于发送电子邮件的 SMTP 电子邮件账户的名称。

Password: SMTP 电子邮件账户的密码。

From: 在发件地址栏显示的电子邮件（通常与 UserName 相同）。

第二：创建一个电子邮件输入页面

接着创建一个输入页面，并将它命名为 Email_Input：

Email_Input.cshtml

```
<!DOCTYPE html>
<html>
<body>
<h1>Request for Assistance</h1>

<form method="post" action="EmailSend.cshtml">
<label>Username:</label>
<input type="text name="customerEmail" />
<label>Details about the problem:</label>
<textarea name="customerRequest" cols="45" rows="4"></textarea>
<p><input type="submit" value="Submit" /></p>
</form>

</body>
</html>
```

输入页面的目的是手机信息，然后提交数据到可以将信息作为电子邮件发送的一个新的页面。

第三：创建一个电子邮件发送页面

接着创建一个用来发送电子邮件的页面，并将它命名为 Email_Send：

Email_Send.cshtml

```
@{ // Read input
var customerEmail = Request["customerEmail"];
var customerRequest = Request["customerRequest"];
try
{
// Send email
WebMail.Send(to:"someone@example.com", subject: "Help request from - " + customerEmail, b
}
catch (Exception ex )
{
<text>@ex</text>
}
}
```

想了解更多关于 ASP.NET Web Pages 应用程序发送电子邮件的信息，请查阅：WebMail 对象参考手册。

ASP.NET Web Pages - PHP

PHP 开发人员请注意，Web Pages 可以用 PHP 编写。

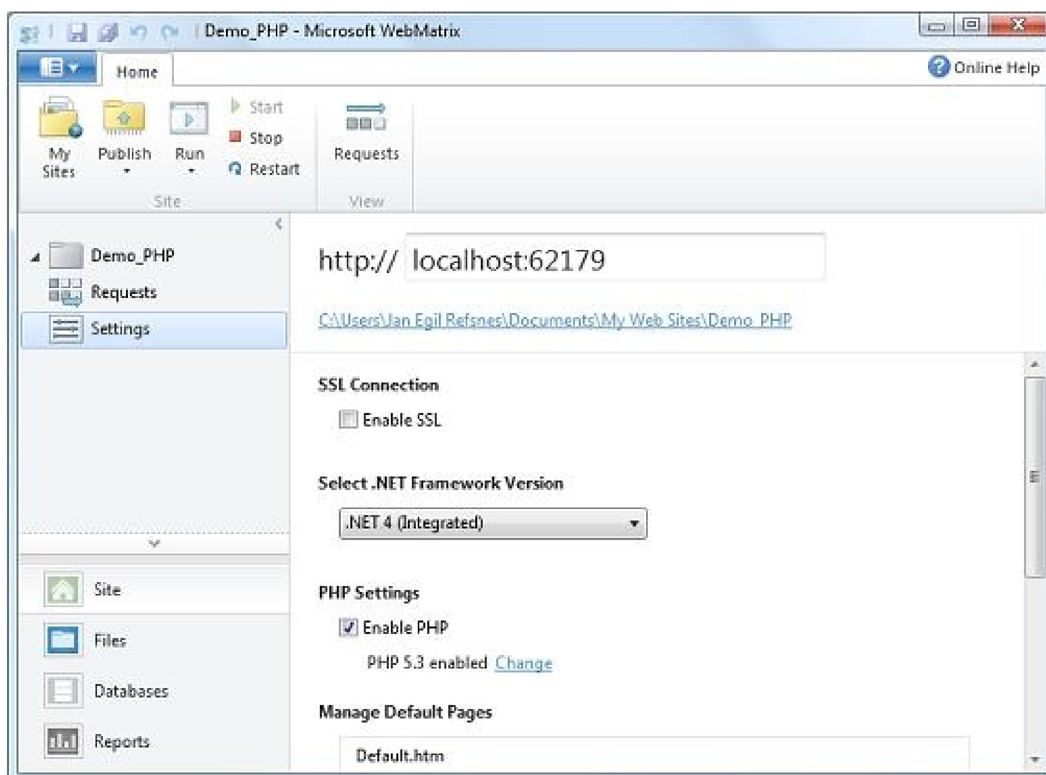
WebMatrix 支持 PHP

乍一看，认为 WebMatrix 只支持微软的技术。这是不正确的。在 WebMatrix 中，您能编写完整的 PHP 应用程序。

创建一个 PHP 站点

在ASP.NET Web Pages - 创建一个网站章节中，您已经创建了一个名为 "Demo" 的空网站，带有一个类型为 "CSHTML" 的空页面。

重复一遍创建的过程，创建一个名为 "Demo_PHP" 的空站点，勾选 "Enable PHP"（如下图所示），创建一个 PHP 类型的空白页，并将它命名 "index.php"，这样您就创建好了您的第一个 PHP 站点。



创建一个 PHP 页面

将下面的代码复制到 "index.php" 文件中：

index.php

```
<!DOCTYPE html>
<html>
<body>

<?php
phpinfo();
?>

</body>
</html>
```

运行文件，看看 PHP 页面的演示。

ASP.NET Web Pages - 发布网站

学习如何在不使用 WebMatrix 的情况下发布 Web Pages 应用程序。

在不使用 WebMatrix 的情况下发布您的应用程序

通过在 WebMatrix（或者 Visual Studio）中使用发布命令，可以发布一个 ASP.NET Web Pages 应用程序到远程服务器上。

此功能会复制所有您的应用程序文件、cshtml页面、图像以及用于 Web Pages、Razor、Helpers、SQL Server Compact（如果使用数据库）所有必需的 DLL 文件。

有时您不想使用 WebMatrix 发布您的应用程序。也许是因为您的托管服务提供商只支持 FTP，也许您已经有一个基于经典 ASP 的网站，也许您想自己复制所有的文件，也许您想使用 Front Page、Expression Web 等其他一些发布软件。

您会遇到问题吗？是的，会的。但是您有办法解决它。

要执行网站复制，您必须知道如何引用正确的文件，哪些 DLL 文件需要复制，并在何处存储它们。

请按照下列步骤操作：

1. 使用最新版本的 ASP.NET

在您继续操作之前，请确保您的主机运行的是最新版的 ASP.NET（4.0 或者 4.5）。

2. 复制 Web 文件夹

从您的开发计算机上复制您的网站（所有文件夹和内容）到远程主机（服务器）上的应用程序文件夹中。

> 如果您的应用程序中包含数据，不要复制数据（详见下面的第 4 点）。

3. 复制 DLL 文件

确保您的远程主机上的 bin 文件夹中包含了和您开发计算机上相同的 dll 文件。

复制 bin 文件夹之后，它应该包含以下文件：

Microsoft.Web.Infrastructure.dll
NuGet.Core.dll
System.Web.Helpers.dll
System.Web.Razor.dll
System.Web.WebPages.Administration.dll
System.Web.WebPages.Deployment.dll
System.Web.WebPages.dll
System.Web.WebPages.Razor.dll
WebMatrix.Data.dll
WebMatrix.WebData

4. 复制您的数据

如果您的应用程序包含数据或者数据库。例如 SQL Server Compact 数据库（在 App_Data 文件夹中的一个 .sdf 文件），请考虑以下几点：

您是否希望发布您的测试数据到远程服务器上？

大多数时候一般是不希望。

如果在您的开发计算机上有测试数据，它将覆盖您的远程主机上的生产数据。

如果您一定要复制 SQL 数据库（.sdf 文件），那么您应该删除数据库中的所有数据，然后从您的开发计算机上复制一个空的 .sdf 文件到服务器上。

就是这样。**GOOD LUCK**！

Razor 教程

ASP.NET Razor - 标记

Razor 不是一种编程语言。它是服务器端的标记语言。

什么是 Razor？

Razor 是一种标记语法，可以让您将基于服务器的代码（Visual Basic 和 C#）嵌入到网页中。

基于服务器的代码可以在网页传送给浏览器时，创建动态 Web 内容。当一个网页被请求时，服务器在返回页面给浏览器之前先执行页面中的基于服务器的代码。通过服务器的运行，代码能执行复杂的任务，比如进入数据库。

Razor 是基于 ASP.NET 的，是为创建 Web 应用程序而设计的。它具有传统 ASP.NET 的功能，但更容易使用并且更容易学习。

Razor 语法

Razor 使用了与 PHP 和经典 ASP 相似的语法。

Razor：

```
<ul>
@for (int i = 0; i < 10; i++) {
<li>@i</li>
}
</ul>
```

PHP：

```
<ul>
<?php
for ($i = 0; $i < 10; $i++) {
echo("<li>$i</li>");
}
?>
</ul>
```

Web Forms（经典 ASP）：

```
<ul>
<% for (int i = 0; i < 10; i++) { %>
<li><% =i %></li>
<% } %>
</ul>
```

Razor 帮助器

ASP.NET 帮助器是通过几行简单的 Razor 代码即可访问的组件。

您可以使用 Razor 语法构建自己的帮助器，或者使用内建的 ASP.NET 帮助器。

下面是一些有用的 Razor 帮助器的简短说明：

- Web Grid（Web 网格）
- Web Graphics（Web 图形）
- Google Analytics（Google 分析）
- Facebook Integration（Facebook 集成）
- Twitter Integration（Twitter 集成）
- Sending Email（发送电子邮件）
- Validation（验证）

Razor 编程语言

Razor 支持 C# (C sharp) 和 VB (Visual Basic)。

ASP.NET Razor - C# 和 VB 代码语法

Razor 同时支持 C# (C sharp) 和 VB (Visual Basic)。

主要的 Razor C# 语法规则

- Razor 代码块包含在 @{ ... } 中
- 内联表达式（变量和函数）以 @ 开头
- 代码语句用分号结束
- 变量使用 var 关键字声明
- 字符串用引号括起来
- C# 代码区分大小写
- C# 文件的扩展名是 .cshtml

C# 实例

```
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Here in Huston it is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

运行实例？

主要的 Razor VB 语法规则

- Razor 代码块包含在 @Code ... End Code 中
- 内联表达式（变量和函数）以 @ 开头
- 变量使用 Dim 关键字声明
- 字符串用引号括起来
- VB 代码不区分大小写
- VB 文件的扩展名是 .vbhtml

实例

```
<!-- Single statement block -->
@Code dim myMessage = "Hello World" End Code

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@Code
dim greeting = "Welcome to our site!"
dim weekDay = DateTime.Now.DayOfWeek
dim greetingMessage = greeting & " Here in Huston it is: " & weekDay
End Code

<p>The greeting is: @greetingMessage</p>
```

运行实例？

它是如何工作的？

Razor 是一种将服务器代码嵌入在网页中的简单的编程语法。

Razor 语法是基于 ASP.NET 框架，专门用于创建 Web 应用程序的部分 Microsoft.NET 框架。

Razor 语法支持所有 ASP.NET 的功能，但是使用的是一种简化语法，对初学者而言更容易学习，对专家而言更有效率的。

Razor 网页可以被描述成带一下两种类型内容的 HTML 网页： HTML 内容和 Razor 代码。

当服务器读取页面时，它首先运行 Razor 代码，然后再发送 HTML 页面到浏览器。在服务器上执行的代码能够执行一些在浏览器上不能完成的任务，比如，访问服务器数据库。服务器代码能创建动态的 HTML 内容，然后发送到浏览器。从浏览器上看，服务器代码生成的 HTML 与静态的 HTML 内容没有什么不同。

带 Razor 语法的 ASP.NET 网页有特殊的文件扩展名 cshtml（Razor C#）或者 vbhtml（Razor VB）。

使用对象

服务器编码往往涉及到对象。

"Date" 对象是一个典型的内置的 ASP.NET 对象，但对象也可以是自定义的，一个网页，一个文本框，一个文件，一个数据库记录，等等。

对象有用于执行的方法。一个数据库记录可能有一个 "Save" 方法，一个图像对象可能有一个 "Rotate" 方法，一个电子邮件对象可能有一个 "Send" 方法，等等。

对象也有用于描述各自特点的属性。一个数据库记录可能有 FirstName 和 LastName 属性。

ASP.NET Date 对象有一个 Now 属性（写成 Date.Now），Now 属性有一个 Day 属性（写成 Date.Now.Day）。下面实例演示了如何访问 Data 对象的一些属性：

实例

```
<table border="1">
<tr>
<th width="100px">Name</th>
<td width="100px">Value</td>
</tr>
<tr>
<td>Day</td><td>@DateTime.Now.Day</td>
</tr>
<tr>
<td>Hour</td><td>@DateTime.Now.Hour</td>
</tr>
<tr>
<td>Minute</td><td>@DateTime.Now.Minute</td>
</tr>
<tr>
<td>Second</td><td>@DateTime.Now.Second</td>
</tr>
</td>
</table>
```

运行实例？

If 和 Else条件

动态网页的一个重要特点是，您可以根据条件决定做什么。

做到这一点的常用方法是使用 if ... else 语句：

实例

```
@{
var txt = "";
if(DateTime.Now.Hour > 12)
{txt = "Good Evening";}
else
{txt = "Good Morning";}
}
<html>
<body>
<p>The message is @txt</p>
</body>
</html>
```

运行实例？

读取用户输入

动态网页的另一个重要特点是，您可以读取用户输入。

输入是通过 Request[] 功能读取的，并且传送输入数据是经过 IsPost 条件判断的：

实例

```
@{
var totalMessage = "";
if(IsPost)
{
var num1 = Request["text1"];
var num2 = Request["text2"];
var total = num1.AsInt() + num2.AsInt();
totalMessage = "Total = " + total;
}
}
<html>
<body style="background-color: beige; font-family: Verdana, Arial;">
<form action="" method="post">
<p><label for="text1">First Number:</label><br>
<input type="text" name="text1" /></p>
<p><label for="text2">Second Number:</label><br>
<input type="text" name="text2" /></p>
<p><input type="submit" value=" Add " /></p>
</form>
<p>@totalMessage</p>
</body>
</html>
```

运行实例？

ASP.NET Razor - C# 变量

变量是用来存储数据的命名实体。

变量

变量是用来存储数据的。

一个变量的名称必须以字母字符开头，并且不能包含空格或者保留字符。

一个变量可以是一个指定的类型，表示它所存储的数据类型。string 变量存储字符串值（"Welcome to W3CSchool.cc"），integer 变量存储数字值（103），date 变量存储日期值，等等。

变量使用 var 关键字声明，或通过使用类型（如果您想声明类型）声明，但是 ASP.NET 通常能自动确定数据类型。

实例

```
// Using the var keyword:
var greeting = "Welcome to W3CSchool.cc";
var counter = 103;
var today = DateTime.Today;

// Using data types:
string greeting = "Welcome to W3CSchool.cc";
int counter = 103;
DateTime today = DateTime.Today;
```

数据类型

下面列出了常用的数据类型：

类型	描述	实例
int	整数（全数字）	103, 12, 5168
float	浮点数	3.14, 3.4e38
decimal	十进制数字（高精度）	1037.196543
bool	布尔值	true, false
string	字符串	"Hello W3CSchool.cc", "John"

运算符

运算符告诉 ASP.NET 在表达式中执行什么样的命令。

C# 语言支持多种运算符。下面列出了常用的运算符：

运算符	描述	实例
=	给一个变量赋值。	`i=6`
+ - * /	加上一个值或者一个变量。 减去一个值或者一个变量。 乘以一个值或者一个变量。 除以一个值或者一个变量。	`i=5+5` `i=5-5` `i=5*5` `i=5/5`
+= -=	变量递增。 变量递减。	`i += 1` `i -= 1`
==	相等。如果值相等则返回 true。	`if (i==10)`
!=	不等。如果值不等则返回 true。	`if (i!=10)`
< > <= >=	小于。 大于。 小于等于。 大于等于。	`if (i<10)` `if (i>10)` `if (i<=10)` if `(i>=10)`
+	连接字符串（一系列互相关联的事物）。	`"w3" + "schools"`
.	点号。分隔对象和方法。	`DateTime.Hour`
()	圆括号。将值进行分组。	`(i+5)`
()	圆括号。传递参数。	`x=Add(i,5)`
[]	中括号。访问数组或者集合的值。	`name[3]`
!	非。真/假取反。	`if (!ready)`
&& ||	逻辑与。 逻辑或。	`if (ready && clear)` `if (ready || clear)`

转换数据类型

从一种数据类型转换到另一种数据类型，有时候是很有用的。

最常见的例子是将字符串输入转换为另一种类型，如整数或者日期。

一般规则下，都是将用户输入看做字符串处理，即使用户输入了数字。因此数值输入必须被转换成数字，然后才能将其用于计算。

下面列出了常用的转换方法：

方法	描述	实例
AsInt() IsInt()	转换字符串为整数。	if (myString.IsInt()) {myInt=myString.AsInt();}
AsFloat() IsFloat()	转换字符串为浮点数。	if (myString.IsFloat()) {myFloat=myString.AsFloat();}
AsDecimal() IsDecimal()	转换字符串为十进制数。	if (myString.IsDecimal()) {myDec=myString.AsDecimal();}
AsDateTime() IsDateTime()	转换字符串为 ASP.NET DateTime 类型。	myString="10/10/2012"; myDate=myString.AsDateTime();
AsBool() IsBool()	转换字符串为布尔值。	myString="True"; myBool=myString.AsBool();
ToString()	转换任何数据类型为字符串。	myInt=1234; myString=myInt.ToString();

ASP.NET Razor - C# 循环和数组

语句在循环中会被重复执行。

For 循环

如果您需要重复执行相同的语句，您可以设定一个循环。

如果您知道要循环的次数，您可以使用 **for** 循环。这种类型的循环在向上计数或向下计数时特别有用：

实例

```
<html>
<body>
@for(var i = 10; i < 21; i++)
{<p>Line @i</p>}
</body>
</html>
```

运行实例？

For Each 循环

如果您使用的是集合或者数组，您会经常用到 **for each** 循环。

集合是一组相似的对象，for each 循环可以遍历集合直到完成。

下面的实例中，遍历 ASP.NET Request.ServerVariables 集合。

实例

```
<html>
<body>
<ul>
@foreach (var x in Request.ServerVariables)
{<li>@x</li>}
</ul>
</body>
</html>
```

运行实例？

While 循环

while 循环是一个通用的循环。

while 循环以 while 关键字开始，后面紧跟着括号，您可以在括号里规定循环将持续多久，然后是重复执行的代码块。

while 循环通常会设定一个递增或者递减的变量用来计数。

下面的实例中，+= 运算符在每执行一次循环时给变量 i 的值加 1。

实例

```
<html>
<body>
@{
var i = 0;
while (i < 5)
{
i += 1;
<p>Line #@i</p>
}
}
</body>
</html>
```

运行实例？

数组

当您要存储多个相似变量但又不想为每个变量都创建一个独立的变量时，可以使用数组来存储：

实例

```
@{
string[] members = {"Jani", "Hege", "Kai", "Jim"};
int i = Array.IndexOf(members, "Kai")+1;
int len = members.Length;
string x = members[2-1];
}
<html>
<body>
<h3>Members</h3>
@foreach (var person in members)
{
<p>@person</p>
}
<p>The number of names in Members are @len</p>
<p>The person at position 2 is @x</p>
<p>Kai is now in position @i</p>
</body>
</html>
```

运行实例？

ASP.NET Razor - C# 逻辑条件

编程逻辑：根据条件执行代码。

If 条件

C# 允许根据条件执行代码。

使用 **if** 语句来判断条件。根据判断结果，if 语句返回 true 或者 false：

- if 语句开始一个代码块
- 条件写在括号里
- 如果条件为真，大括号内的代码被执行

实例

```
@{var price=50;}
<html>
<body>
@if (price>30)
{
<p>The price is too high.</p>
}
</body>
</html>
```

运行实例 ?

Else 条件

if 语句可以包含 **else** 条件。

else 条件定义了当条件为假时被执行的代码。

实例

```
@{var price=20;}
<html>
<body>
@if (price>30)
{
<p>The price is too high.</p>
}
else
{
<p>The price is OK.</p>
}
</body>
</html>
```

运行实例？

注释：在上面的实例中，如果第一个条件为真，if 块的代码将会被执行。else 条件覆盖了除 if 条件之外的"其他所有情况"。

Else If 条件

多个条件判断可以使用 **else if** 条件：

实例

```
@{var price=25;}
<html>
<body>
@if (price>=30)
{
<p>The price is high.</p>
}
else if (price>20 && price<30)
{
<p>The price is OK.</p>
}
else
{
<p>The price is low.</p>
}
</body>
</html>
```

运行实例？

在上面的实例中，如果第一个条件为真，if 块的代码将会被执行。

如果第一个条件不为真且第二个条件为真，else if 块的代码将会被执行。

else if 条件的数量不受限制。

如果 if 和 else if 条件都不为真，最后的 else 块（不带条件）覆盖了"其他所有情况"。

Switch 条件

switch 块可以用来测试一些单独的条件：

实例

```
@{
var weekday=DateTime.Now.DayOfWeek;
var day=weekday.ToString();
var message="";
}
<html>
<body>
@switch(day)
{
case "Monday":
message="This is the first weekday.";
break;
case "Thursday":
message="Only one day before weekend.";
break;
case "Friday":
message="Tomorrow is weekend!";
break;
default:
message="Today is " + day;
break;
}
<p>@message</p>
</body>
</html>
```

运行实例？

测试值（day）是写在括号中。每个单独的测试条件都有一个以分号结束的 case 值和以 break 语句结束的任意数量的代码行。如果测试值与 case 值相匹配，相应的代码行被执行。

switch 块有一个默认的情况（default:），当所有的指定的情况都不匹配时，它覆盖了"其他所有情况"。

ASP.NET Razor - VB 变量

变量是用来存储数据的命名实体。

变量

变量是用来存储数据的。

一个变量的名称必须以字母字符开头，并且不能包含空格或者保留字符。

一个变量可以是一个指定的类型，表示它所存储的数据类型。string 变量存储字符串值（"Welcome to W3CSchool.cc"），integer 变量存储数字值（103），date 变量存储日期值，等等。

变量使用 Dim 关键字声明，或通过使用类型（如果您想声明类型）声明，但是 ASP.NET 通常能自动确定数据类型。

实例

```
// Using the Dim keyword:
Dim greeting = "Welcome to W3CSchool.cc"
Dim counter = 103
Dim today = DateTime.Today

// Using data types:
Dim greeting As String = "Welcome to W3CSchool.cc"
Dim counter As Integer = 103
Dim today As DateTime = DateTime.Today
```

数据类型

下面列出了常用的数据类型：

类型	描述	实例
integer	整数（全数字）	103, 12, 5168
double	64 位浮点数	3.14, 3.4e38
decimal	十进制数字（高精度）	1037.196543
boolean	布尔值	true, false
string	字符串	"Hello W3CSchool.cc", "John"

运算符

运算符告诉 ASP.NET 在表达式中执行什么样的命令。

VB 语言支持多种运算符。下面列出了常用的运算符：

运算符	描述	实例
=	给一个变量赋值。	`i=6`
+ - * /	加上一个值或者一个变量。 减去一个值或者一个变量。 乘以一个值或者一个变量。 除以一个值或者一个变量。	`i=5+5` `i=5-5` `i=5*5` `i=5/5`
+= -=	变量递增。 变量递减。	`i += 1` `i -= 1`
=	相等。如果值相等则返回 true。	`if i=10`
<>	不等。如果值不等则返回 true。	`if <>10`
< > <= >=	小于。 大于。 小于等于。 大于等于。	`if i<10` `if i>10` `if i<=10` `if i>=10`
&	连接字符串（一系列互相关联的事物）。	`"w3" & "schools"`
.	点号。分隔对象和方法。	`DateTime.Hour`
()	圆括号。将值进行分组。	`(i+5)`
()	圆括号。传递参数。	`x=Add(i,5)`
()	圆括号。访问数组或者集合的值。	`name(3)`
Not	非。真/假取反。	`if Not ready`
And OR	逻辑与。 逻辑或。	`if ready And clear` `if ready Or clear`
AndAlso orElse	扩展的逻辑与。 扩展的逻辑或。	`if ready AndAlso clear` `if ready OrElse clear`

转换数据类型

从一种数据类型转换到另一种数据类型，有时候是很有用的。

最常见的例子是将字符串输入转换为另一种类型，如整数或者日期。

一般规则下，都是将用户输入看做字符串处理，即使用户输入了数字。因此数值输入必须被转换成数字，然后才能将其用于计算。

下面列出了常用的转换方法：

方法	描述	实例
AsInt() IsInt()	转换字符串为整数。	if myString.IsInt() then myInt=myString.AsInt() end if
AsFloat() IsFloat()	转换字符串为浮点数。	if myString.IsFloat() then myFloat=myString.AsFloat() end if
AsDecimal() IsDecimal()	转换字符串为十进制数。	if myString.IsDecimal() then myDec=myString.AsDecimal() end if
AsDateTime() IsDateTime()	转换字符串为 ASP.NET DateTime 类型。	myString="10/10/2012" myDate=myString.AsDateTime()
AsBool() IsBool()	转换字符串为布尔值。	myString="True" myBool=myString.AsBool()
ToString()	转换任何数据类型为字符串。	myInt=1234 myString=myInt.ToString()

ASP.NET Razor - VB 循环和数组

语句在循环中会被重复执行。

For 循环

如果您需要重复执行相同的语句，您可以设定一个循环。

如果您知道要循环的次数，您可以使用 **for** 循环。这种类型的循环在向上计数或向下计数时特别有用：

实例

```
<html>
<body>
@For i=10 To 21
@<p>Line #@i</p>
Next i
</body>
</html>
```

运行实例 ?

For Each 循环

如果您使用的是集合或者数组，您会经常用到 **for each** 循环。

集合是一组相似的对象，for each 循环可以遍历集合直到完成。

下面的实例中，遍历 ASP.NET Request.ServerVariables 集合。

实例

```
<html>
<body>
<ul>
@For Each x In Request.ServerVariables
@<li>@x</li>
Next x
</ul>
</body>
</html>
```

运行实例？

While 循环

while 循环是一个通用的循环。

while 循环以 while 关键字开始，后面紧跟着括号，您可以在括号里规定循环将持续多久，然后是重复执行的代码块。

while 循环通常会设定一个递增或者递减的变量用来计数。

下面的实例中，+= 运算符在每执行一次循环时给变量 i 的值加 1。

实例

```
<html>
<body>
@Code
Dim i=0
Do While i<5
i += 1
@<p>Line #@i</p>
Loop
End Code
</body>
</html>
```

运行实例？

数组

当您要存储多个相似变量但又不想为每个变量都创建一个独立的变量时，可以使用数组来存储：

实例

```
@Code
Dim members As String()={"Jani","Hege","Kai","Jim"}
i=Array.IndexOf(members,"Kai")+1
len=members.Length
x=members(2-1)
end Code
<html>
<body>
<h3>Members</h3>
@For Each person In members
@<p>@person</p>
Next person
<p>The number of names in Members are @len</p>
<p>The person at position 2 is @x</p>
<p>Kai is now in position @i</p>
</body>
</html>
```

运行实例？

ASP.NET Razor - VB 逻辑条件

编程逻辑：根据条件执行代码。

If 条件

VB 允许根据条件执行代码。

使用 **if** 语句来判断条件。根据判断结果，if 语句返回 true 或者 false：

- if 语句开始一个代码块
- 条件写在 if 和 then 之间
- 如果条件为真，if ... then 和 end if 之间的代码被执行

实例

```
@Code
Dim price=50
End Code
<html>
<body>
@If price>30 Then
@<p>The price is too high.</p>
End If
</body>
</html>
```

运行实例？

Else 条件

if 语句可以包含 **else** 条件。

else 条件定义了当条件为假时被执行的代码。

实例

```
@Code
Dim price=20
End Code
<html>
<body>
@if price>30 then
@<p>The price is too high.</p>
Else
@<p>The price is OK.</p>
End If
</body>
</htmlV>
```

运行实例？

注释：在上面的实例中，如果第一个条件为真，if 块的代码将会被执行。else 条件覆盖了除 if 条件之外的"其他所有情况"。

ElseIf 条件

多个条件判断可以使用 **elseif** 条件：

实例

```
@Code
Dim price=25
End Code
<html>
<body>
@If price>=30 Then
@<p>The price is high.</p>
ElseIf price>20 And price<30
@<p>The price is OK.</p>
Else
@<p>The price is low.</p>
End If
</body>
</html>
```

运行实例？

在上面的实例中，如果第一个条件为真，if 块的代码将会被执行。

如果第一个条件不为真且第二个条件为真，elseif 块的代码将会被执行。

elseif 条件的数量不受限制。

如果 if 和 elseif 条件都不为真，最后的 else 块（不带条件）覆盖了"其他所有情况"。

Select 条件

select 块可以用来测试一些单独的条件：

实例

```
@Code
Dim weekday=DateTime.Now.DayOfWeek
Dim day=weekday.ToString()
Dim message=""
End Code
<html>
<body>
@Select Case day
Case "Monday"
message="This is the first weekday."
Case "Thursday"
message="Only one day before weekend."
Case "Friday"
message="Tomorrow is weekend!"
Case Else
message="Today is " & day
End Select
<p>@message</p>
</body>
</html>
```

运行实例？

"Select Case" 后面紧跟着测试值（day）。每个单独的测试条件都有一个 case 值和任意数量的代码行。如果测试值与 case 值相匹配，相应的代码行被执行。

select 块有一个默认的情况（Case Else），当所有的指定的情况都不匹配时，它覆盖了"其他所有情况"。

MVC 教程

ASP.NET MVC 教程

ASP.NET 是一个使用 HTML、CSS、JavaScript 和服务器脚本创建网页和网站的开发框架。

ASP.NET 支持三种不同的开发模式：
Web Pages（Web 页面）、MVC（Model View Controller 模型-视图-控制器）、Web Forms（Web 窗体）。

本教程介绍 **MVC**。

Web Pages
MVC
Web Forms

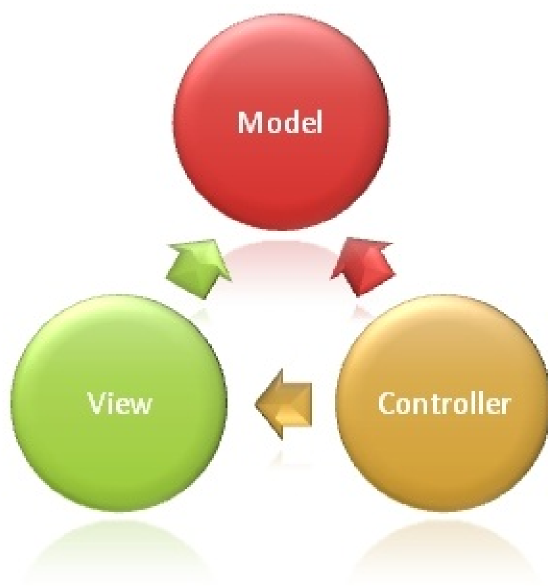
MVC 编程模式

MVC 是三种 ASP.NET 编程模式中的一种。

MVC 是一种使用 MVC（Model View Controller 模型-视图-控制器）设计创建 Web 应用程序的模式：

- Model（模型）表示应用程序核心（比如数据库记录列表）。
- View（视图）显示数据（数据库记录）。
- Controller（控制器）处理输入（写入数据库记录）。

MVC 模式同时提供了对 HTML、CSS 和 JavaScript 的完全控制。



MVC 模式定义 Web 应用程序 带有三个逻辑层： 业务层（模型逻辑） 显示层（视图逻辑） 输入控制（控制器逻辑）

Model（模型）是应用程序中用于处理应用程序数据逻辑的部分。
通常模型对象负责在数据库中存取数据。

View（视图）是应用程序中处理数据显示的部分。
通常视图是依据模型数据创建的。

Controller（控制器）是应用程序中处理用户交互的部分。
通常控制器负责从视图读取数据，控制用户输入，并向模型发送数据。

MVC 分层有助于管理复杂的应用程序，因为您可以在一个时间内专门关注一个方面。例如，您可以在不依赖业务逻辑的情况下专注于视图设计。同时也让应用程序的测试更加容易。

MVC 分层同时也简化了分组开发。不同的开发人员可同时开发视图、控制器逻辑和业务逻辑。

Web Forms 对比 MVC

MVC 编程模式是对传统 ASP.NET（Web Forms）的一种轻量级的替代方案。它是轻量级的、可测试性高的框架，同时整合了所有已有的 ASP.NET 特性，比如母版页、安全性和认证。

Visual Studio Express 2012/2010

Visual Studio Express 是 Microsoft Visual Studio 的免费版本。

Visual Studio Express 是为 MVC（和 Web Forms）量身定制的开发工具。

Visual Studio Express 包含：

- MVC 和 Web Forms
- 拖拽 Web 控件和 Web 组件
- Web 服务器语言（Razor 使用 VB 或者 C#）
- Web 服务器（IIS Express）
- 数据库服务器（SQL Server Compact）
- 完整的 Web 开发框架（ASP.NET）

如果您已经安装了 Visual Studio Express，您将从本教程中学到更多。

如果您想安装 Visual Studio Express，请点击下列链接中的一个：

Visual Web Developer 2012（Windows 7 或者 Windows 8）

Visual Web Developer 2010（Windows Vista 或者 XP）

> 在您首次安装完 Visual Studio Express 之后，您可以通过再次运行安装程序来安装补丁和服务包，只需要再次点击链接即可。

ASP.NET MVC 参考手册

在本教程的最后，我们提供了完整的 ASP.NET MVC 参考手册供您查阅。

ASP.NET MVC - Internet 应用程序

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 1 部分：创建应用程序。

我们将构建什么

我们将构建一个支持添加、编辑、删除和列出数据库存储信息的 Internet 应用程序。

我们将做什么

Visual Web Developer 提供了构建 Web 应用程序的不同模板。

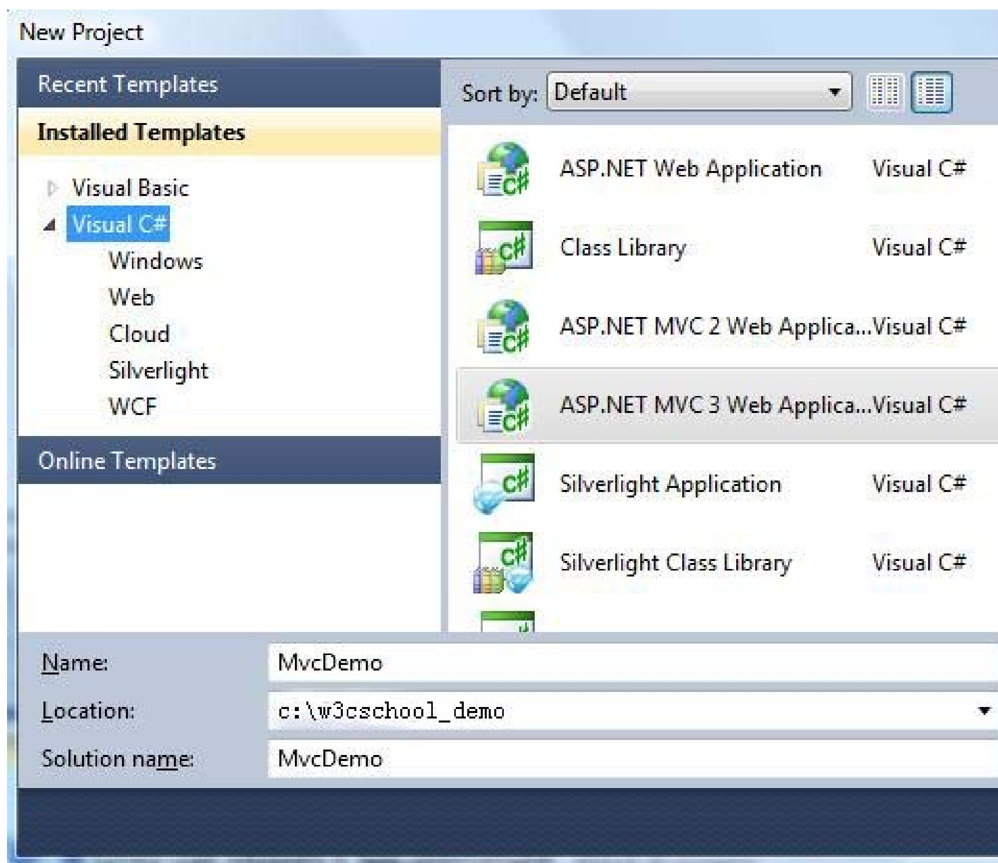
我们将使用 **Visual Web Developer** 来创建一个带 **HTML5** 标记的空的 MVC Internet 应用程序。

当这个空白的 Internet 应用程序被创建之后，我们将逐步向该应用添加代码，直到全部完成。我们将使用 **C#** 作为编程语言，并使用最新的 **Razor** 服务器代码标记。

沿着这个思路，我们将讲解这个应用程序的内容、代码和所有组件。

创建 **Web** 应用程序

如果您已经安装了 Visual Web Developer，请启动 Visual Web Developer 并选择 **New Project** 来新建项目。 否则您就只能通过阅读教程来学习了。



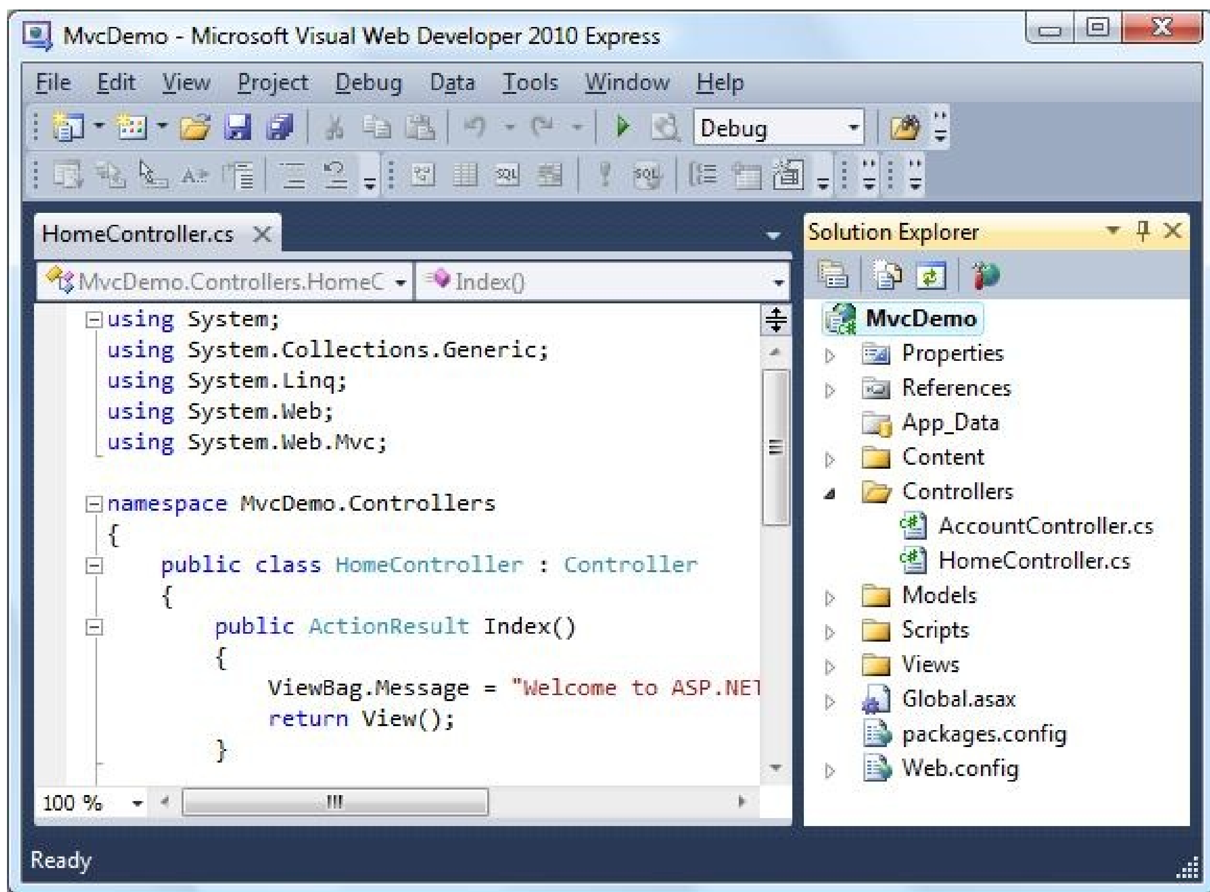
在 New Project 对话框中：

- 打开**Visual C#**模板
- 选择模板 **ASP.NET MVC 3 Web Application**
- 设置项目名称为 **MvcDemo**
- 设置磁盘位置，比如 **c:\w3cschool_demo**
- 点击 **OK**

当 New Project 对话框打开时：

- 选择 **Internet Application** 模板
- 选择 **Razor Engine**（Razor 引擎）
- 选择 **HTML5 Markup**（HTML5 标记）
- 点击 **OK**

Visual Studio Express 将创建一个如下所示的类似项目：



我们将在本教程的下一章中探究有关文件和文件夹的内容。

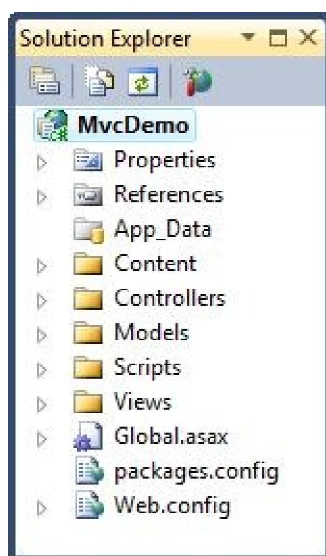
ASP.NET MVC - 应用程序文件夹

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 2 部分：探究应用程序文件夹。

MVC 文件夹

一个典型的 ASP.NET MVC Web 应用程序的文件夹内容如下所示：



应用程序信息

Properties
References

应用程序文件夹

App_Data 文件夹
Content 文件夹
Controllers 文件夹
Models 文件夹
Scripts 文件夹
Views 文件夹

配置文件

Global.asax
packages.config
Web.config

所有的 MVC 应用程序的文件夹名称都是相同的。MVC 框架是基于默认的命名。控制器写在 Controllers 文件夹中，视图写在 Views 文件夹中，模型写在 Models 文件夹中。您不必再应用程序代码中使用文件夹名称。

标准化的命名减少了代码量，同时有利于开发人员对 MVC 项目的理解。

下面是对每个文件夹内容的简短概述：

App_Data 文件夹

App_Data 文件夹用于存储应用程序数据。

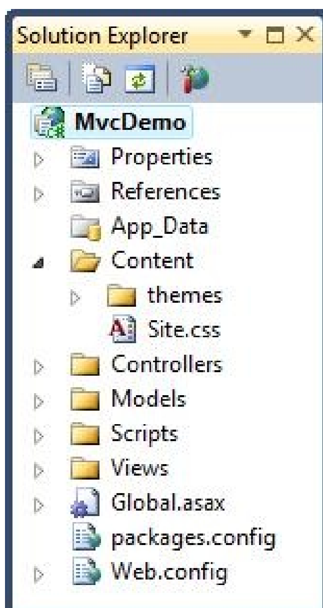
我们将在本教程后面的章节中介绍添加 SQL 数据库到 App_Data 文件夹。

Content 文件夹

Content 文件夹用于存放静态文件，比如样式表（CSS 文件）、图标和图像。

Visual Web Developer 会自动添加一个 **themes** 文件夹到 Content 文件夹中。themes 文件夹存放 jQuery 样式和图片。在项目中，您可以删除这个 themes 文件夹。

Visual Web Developer 同时也会添加一个标准的样式表文件到项目中：即 content 文件夹中的 **Site.css** 文件。这个样式表文件是您想要改变应用程序样式时需要编辑的文件。



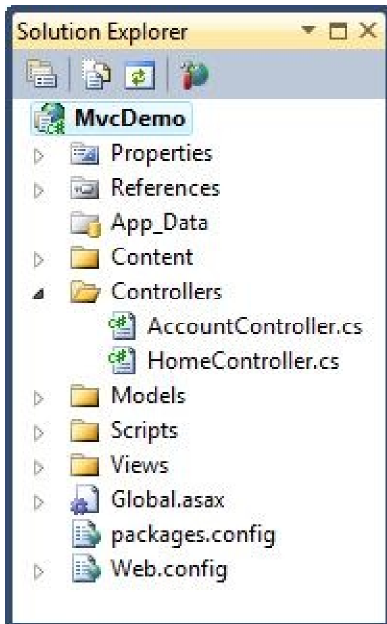
我们将在本教程的下一章中编辑这个样式表文件（Site.css）。

Controllers 文件夹

Controllers 文件夹包含负责处理用户输入和相应的控制器类。

MVC 要求所有控制器文件的名称以 "Controller" 结尾。

Visual Web Developer 已经创建好一个 Home 控制器（用于 Home 页面和 About 页面）和一个 Account 控制器（用于 Login 页面）：



我们将在本教程后面的章节中创建更多的控制器。

Models 文件夹

Models 文件夹包含表示应用程序模型的类。模型控制并操作应用程序的数据。

我们将在本教程后面的章节中创建模型（类）。

Views 文件夹

Views 文件夹用于存储与应用程序的显示相关的 HTML 文件（用户界面）。

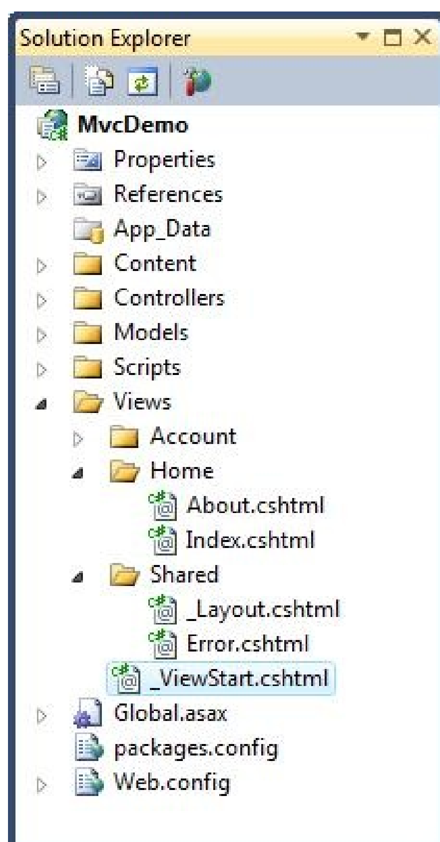
Views 文件夹中包含每个控制器对应的一个文件夹。

在 Views 文件夹中，Visual Web Developer 已经创建了一个 Account 文件夹、一个 Home 文件夹、一个 Shared 文件夹。

Account 文件夹包含用于用户账号注册和登录的页面。

Home 文件夹用于存储诸如 home 页和 about 页之类的应用程序页面。

Shared 文件夹用于存储控制器间分享的视图（母版页和布局页）。

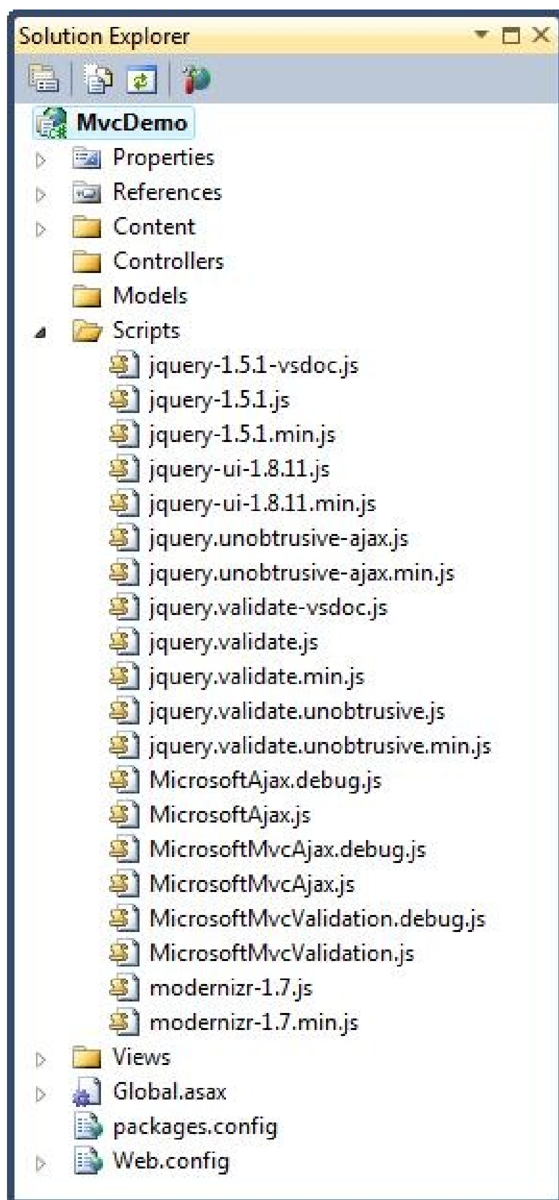


我们将在本教程的下一章中编辑这些布局文件。

Scripts 文件夹

Scripts 文件夹存储应用程序的 JavaScript 文件。

默认情况下，Visual Web Developer 在这个文件夹中存放标准的 MVC、Ajax 和 jQuery 文件：



注释：名为 "modernizr" 的文件时用于在应用程序中支持 HTML5 和 CSS3 的 JavaScript 文件。

ASP.NET MVC - 样式和布局

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 3 部分：添加样式和统一的外观（布局）。

添加布局

文件 _Layout.cshtml 表示应用程序中每个页面的布局。它位于 Views 文件夹中的 Shared 文件夹。

打开文件 _Layout.cshtml，把内容替换成：

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>@ViewBag.Title</title>
<link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />
<script src="@Url.Content("~/Scripts/jquery-1.5.1.min.js")"></script>
<script src="@Url.Content("~/Scripts/modernizr-1.7.min.js")"></script>
</head>
<body>
<ul id="menu">
<li>@Html.ActionLink("Home", "Index", "Home")</li>
<li>@Html.ActionLink("Movies", "Index", "Movies")</li>
<li>@Html.ActionLink("About", "About", "Home")</li>
</ul>
<section id="main">
@RenderBody()
<p>Copyright W3CSchool 2012\. All Rights Reserved.</p>
</section>
</body>
</html>
```

HTML 帮助器

在上面的代码中，HTML 帮助器用于修改 HTML 输出：

```
@Url.Content() - URL 内容将在此处插入。

@Html.ActionLink() - HTML 链接将在此处插入。
```

在本教程后面的章节中，您将学到更多关于 HTML 帮助器的知识。

Razor 语法

在上面的代码中，红色标记的代码是使用 Razor 标记的 C#。

```
@ViewBag.Title - 页面标题将在此处插入。

@RenderBody() - 页面内容将在此处呈现。
```

您可以在我们的 Razor 教程中学习关于 C# 和 VB（Visual Basic）的 Razor 标记的知识。

添加样式

应用程序的样式表是 Site.css，位于 Content 文件夹中。

打开文件 Site.css，把内容替换成：

```
body
{
font: "Trebuchet MS", Verdana, sans-serif;
background-color: #5c87b2;
color: #696969;
}
h1
{
border-bottom: 3px solid #cc9900;
font: Georgia, serif;
color: #996600;
}
#main
{
padding: 20px;
background-color: #ffffff;
border-radius: 0 4px 4px 4px;
}
a
{
color: #034af3;
}
/* Menu Styles ------------------------------*/
ul#menu
{
padding: 0px;
position: relative;
margin: 0;
}
ul#menu li
{
display: inline;
}
ul#menu li a
{
background-color: #e8eef4;
padding: 10px 20px;
text-decoration: none;
line-height: 2.8em;
/*CSS3 properties*/
border-radius: 4px 4px 0 0;
}
ul#menu li a:hover
{
background-color: #ffffff;
}
/* Forms Styles ------------------------------*/
fieldset
{
padding-left: 12px;
}
```

```
fieldset label
{
display: block;
padding: 4px;
}
input[type="text"], input[type="password"]
{
width: 300px;
}
input[type="submit"]
{
padding: 4px;
}
/* Data Styles ------------------------------*/
table.data
{
background-color:#ffffff;
border:1px solid #c3c3c3;
border-collapse:collapse;
width:100%;
}
table.data th
{
background-color:#e8eef4;
border:1px solid #c3c3c3;
padding:3px;
}
table.data td
{
border:1px solid #c3c3c3;
padding:3px;
}
```

_ViewStart 文件

Shared 文件夹（位于 Views 文件夹内）中的 _ViewStart 文件包含如下内容：

```
@{Layout = "~/Views/Shared/_Layout.cshtml";}
```

这段代码被自动添加到由应用程序显示的所有视图。

如果您删除了这个文件，则必须向所有视图中添加这行代码。

在本教程后面的章节中，您将学到更多关于视图的知识。

ASP.NET MVC - 控制器

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

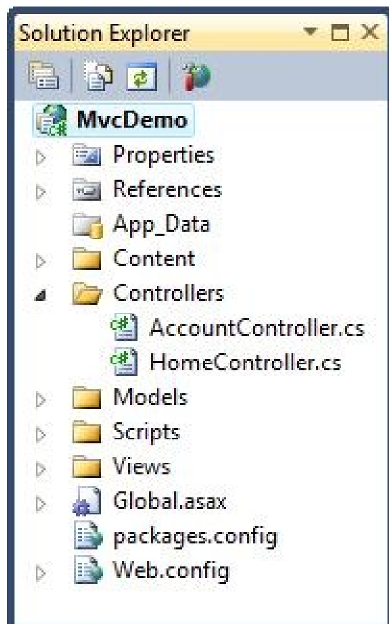
第 4 部分：添加控制器。

Controllers 文件夹

Controllers 文件夹包含负责处理用户输入和响应的控制类。

MVC 要求所有控制器文件的名称以 "Controller" 结尾。

在我们的实例中，Visual Web Developer 已经创建好了一下文件： **HomeController.cs**（用于 Home 页面和 About 页面）和**AccountController.cs** （用于登录页面） ：



Web 服务器通常会将进入的 URL 请求直接映射到服务器上的磁盘文件。例如：URL 请求 "http://www.w3cschool.cc/index.php" 将直接映射到服务器根目录上的文件 "index.php"。

MVC 框架的映射方式有所不同。MVC 将 URL 映射到方法。这些方法在类中被称为"控制器"。

控制器负责处理进入的请求，处理输入，保存数据，并把响应发送回客户端。

Home 控制器

在我们应用程序中的控制器文件**HomeController.cs**，定义了两个控件 **Index** 和 **About**。

把 HomeController.cs 文件的内容替换成：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MvcDemo.Controllers
{
public class HomeController : Controller
{
public ActionResult Index()
{return View();}

public ActionResult About()
{return View();}
}
}
```

Controller 视图

Views 文件夹中的文件 **Index.cshtml** 和 **About.cshtml** 定义了控制器中的 ActionResult 视图 Index() 和 About()。

ASP.NET MVC - 视图

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 5 部分：添加用于显示应用程序的视图。

Views 文件夹

Views 文件夹存储的是与应用程序显示（用户界面）相关的文件（HTML 文件）。根据所采用的语言内容，这些文件可能扩展名可能是 html、asp、aspx、cshtml 和 vbhtml。

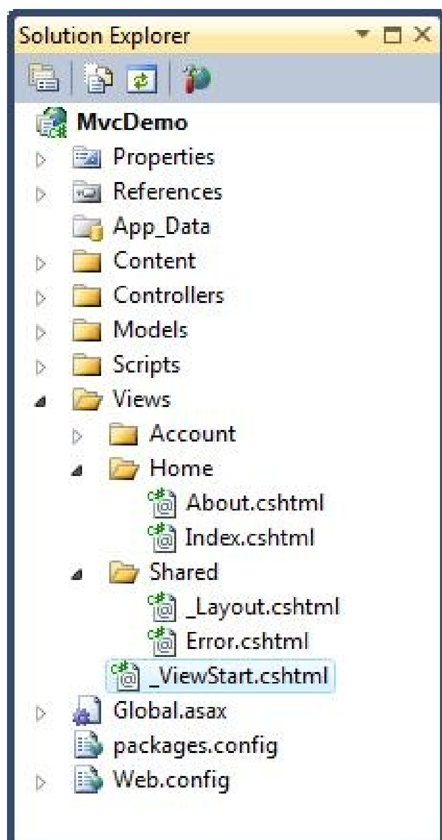
Views 文件夹中包含每个控制器对应的一个文件夹。

在 Views 文件夹中，Visual Web Developer 已经创建了一个 Account 文件夹、一个 Home 文件夹、一个 Shared 文件夹。

Account 文件夹包含用于用户账号注册和登录的页面。

Home 文件夹用于存储诸如 home 页和 about 页之类的应用程序页面。

Shared 文件夹用于存储控制器间分享的视图（母版页和布局页）。



ASP.NET 文件类型

在 Views 文件夹中可以看到以下 HTML 文件类型：

文件类型	扩展名
纯 HTML	.htm or .html
经典 ASP	.asp
经典 ASP.NET	.aspx
ASP.NET Razor C#	.cshtml
ASP.NET Razor VB	.vbhtml

Index 文件

文件 Index.cshtml 表示应用程序的 Home 页面。它是应用程序的默认文件（首页文件）。

在文件中写入以下内容：

```
@{ViewBag.Title = "Home Page";}

<h1>Welcome to W3CSchool.cc</h1>

<p>Put Home Page content here</p>
```

About 文件

文件 About.cshtml 表示应用程序的 About 页面。

在文件中写入以下内容：

```
@{ViewBag.Title = "About Us";}

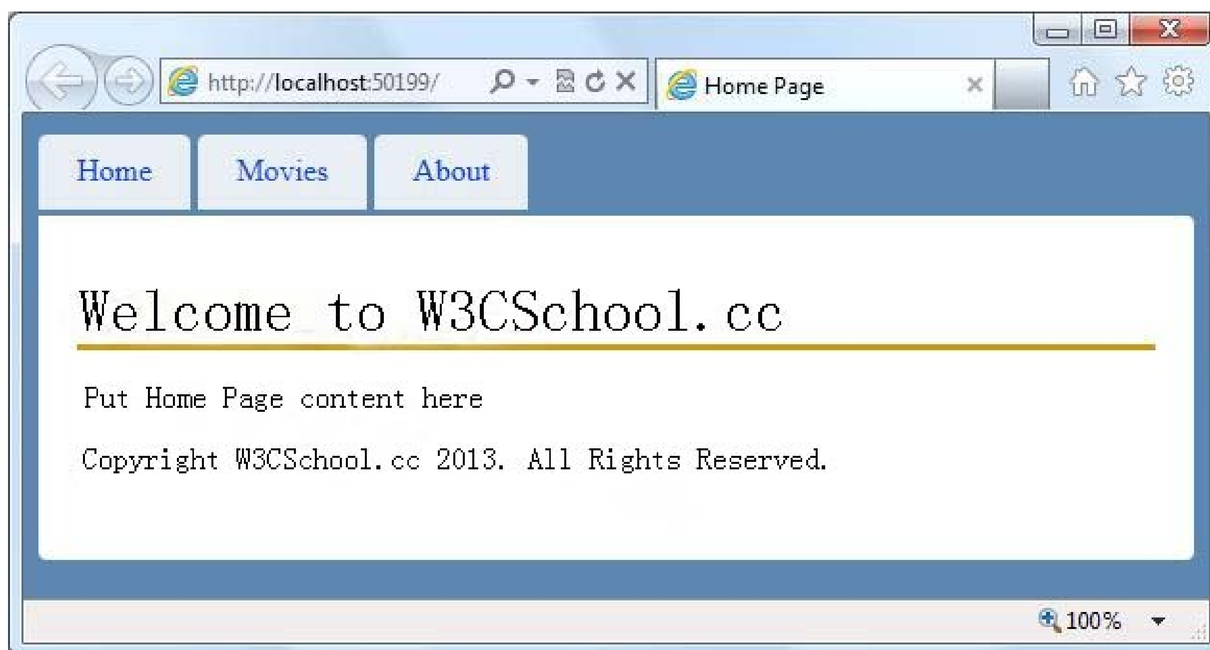
<h1>About Us</h1>

<p>Put About Us content here</p>
```

运行应用程序

选择 Debug，从 Visual Web Developer 菜单中启动调试 Start Debugging（或者按 F5）。

您的应用程序将显示如下：



点击 "Home" 标签页和 "About" 标签页，看看它是如何运作的。

祝贺您

祝贺您。您已经创建好了您的第一个 MVC 应用程序。

注释：您暂时还不能点击 "Movies" 标签页。我们将在本教程的后面章节中为 "Movies" 标签页添加代码。

ASP.NET MVC - SQL 数据库

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 6 部分：添加数据库。

创建数据库

Visual Web Developer 带有名为 SQL Server Compact 免费的 SQL 数据库。

本教程所需的这个数据库可以通过以下几个简单的步骤来创建：

- 右击 **Solution Explorer** 窗口中的 **App_Data** 文件夹
- 选择 **Add, New Item**
- 选择 **SQL Server Compact Local Database** *
- 将数据库命名为 **Movies.sdf**
- 点击 **Add** 按钮

* 如果选项中没有 SQL Server Compact Local Database，则说明您尚未在计算机上安装 SQL Server Compac。请通过以下链接进行安装：SQL Server Compact

Visual Web Developer 会自动在 App_Data 文件夹中创建该数据库。

注释：在本教程中，需要您掌握一些关于 SQL 数据库的基础知识。如果您想先学习这个主题，请访问我们的 SQL 教程。

添加数据库表

双击 **App_Data** 文件夹中的 **Movies.sdf** 文件，将打开 **Database Explorer** 窗口。

如需在数据库中创建一个新的表，请右击 **Tables** 文件夹，然后选择 **Create Table**。

创建如下的列：

列	类型	是否允许为 **Null**
ID	int (primary key)	No
Title	nvarchar(100)	No
Director	nvarchar(100)	No
Date	datetime	No

对列的解释：

ID 是用于标识表中每条记录的整数（全数字）。

Title 是 100 个字符长度的文本列，用于存储影片的名称。

Director 是 100 个字符长度的文本列，用于存储导演的名字。

Date 是日期列，用于存储影片的发布日期。

在创建好上述列之后，您必须将 ID 列设置为表的主键（记录标识符）。要做到这点，请点击列名（ID），并选择 **Primary Key**。在 **Column Properties** 窗口中，设置 **Identity** 属性为 **True**：

Name:

Column Name	Data Type	Len...	Allow Nulls	Unique	Primary Key
ID	int	4	No	Yes	Yes
Title	nvarchar	100	No	No	No
Director	nvarchar	100	No	No	No
Date	datetime	8	No	No	No

Delete

Default Value	
Identity	**True**
Identity Increment	**1**
Identity Seed	**1**
Is RowGuid	False
Precision	
Scale	

当您创建好表列后，保存表并命名为 **MovieDBs**。

注释：

我们特意把表命名为 "MovieDBs"（以 s 结尾）。在下一章中，您将看到用于数据模型的 "MovieDB"。这看起来有点奇怪，不过这种命名惯例能确保控制器连接上数据库表，您必须这么使用。

添加数据库记录

您可以使用 Visual Web Developer 向 movie 数据库中添加一些测试记录。

双击 **App_Data** 文件夹中的 **Movies.sdf** 文件。

右击 Database Explorer 窗口中的 **MovieDBs** 表，并选择 **Show Table Data**。

添加一些记录：

ID	Title	Director	Date
1	Psycho	Alfred Hitchcock	01.01.1960
2	La Dolce Vita	Federico Fellini	01.01.1960

注释：ID 列会自动更新，您可以不用编辑它。

添加连接字符串

向您的 **Web.config** 文件中的 **<connectionStrings>** 元素添加如下元素：

ASP.NET MVC - 模型

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

第 7 部分：添加数据模型。

MVC 模型

MVC 模型包含了除纯视图和控制器逻辑以外的其他所有应用程序逻辑（业务逻辑、验证逻辑、数据访问逻辑）。

通过 MVC，模型可以控制并操作应用程序数据。

Models 文件夹

Models 文件夹包含表示应用程序模型的类。

Visual Web Developer 自动创建一个 **AccountModels.cs** 文件，该文件包含用于应用程序安全的模型。

AccountModels 包含 **LogOnModel**、**ChangePasswordModel** 和 **RegisterModel**。

添加数据库模型

本教程所需的数据库模型可以通过以下几个简单的步骤来创建：

- 在 **Solution Explorer**窗口中，右击 **Models** 文件夹，并选择 **Add** 和 **Class**。
- 将类命名为 **MovieDB.cs**，然后点击 **Add**。
- 编辑这个类：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Entity;

namespace MvcDemo.Models
{
public class MovieDB
{
public int ID { get; set; }
public string Title { get; set; }
public string Director { get; set; }
public DateTime Date { get; set; }

}
public class MovieDBContext : DbContext
{
public DbSet<MovieDB> Movies { get; set; }
}
}
```

注释：

我们特意把模型命名为 "MovieDB"。在上一章中，您已经看到用于数据库表的 "MovieDBs"（以 s 结尾）。这看起来有点奇怪，不过这种命名惯例能确保模型连接上数据库表，您必须这么使用。

添加数据库控制器

本教程所需的数据库控制器可以通过以下几个简单的步骤来创建：

- 重建您的项目：选择 **Debug**，然后从菜单中选择 **Build MvcDemo**。
- 在 Solution Explorer（解决方案资源管理器）中，右击 **Controllers** 文件夹，选择 **Add** 和 **Controller**。
- 设置控制器名称为 **MoviesController**。
- 选择模板：**Controller with read/write actions and views, using Entity Framework**
- 选择模型类：**MovieDB (MvcDemo.Models)**
- 选择 data context 类：**MovieDBContext (MvcDemo.Models)**
- 选择视图 **Razor (CSHTML)**
- 点击 **Add**

Visual Web Developer 将创建以下文件：

- **Controllers** 文件夹中的 **MoviesController.cs** 文件
- **Views** 文件夹中的 **Movies** 文件夹

添加数据库视图

在 Movies 文件夹中，会自动创建以下文件：

- Create.cshtml
- Delete.cshtml
- Details.cshtml
- Edit.cshtml
- Index.cshtml

祝贺您

祝贺您。您已经向应用程序添加了您的第一个 MVC 数据模型。

现在您可以点击 "Movies" 标签页了。

ASP.NET MVC - 安全

为了学习 ASP.NET MVC，我们将构建一个 Internet 应用程序。

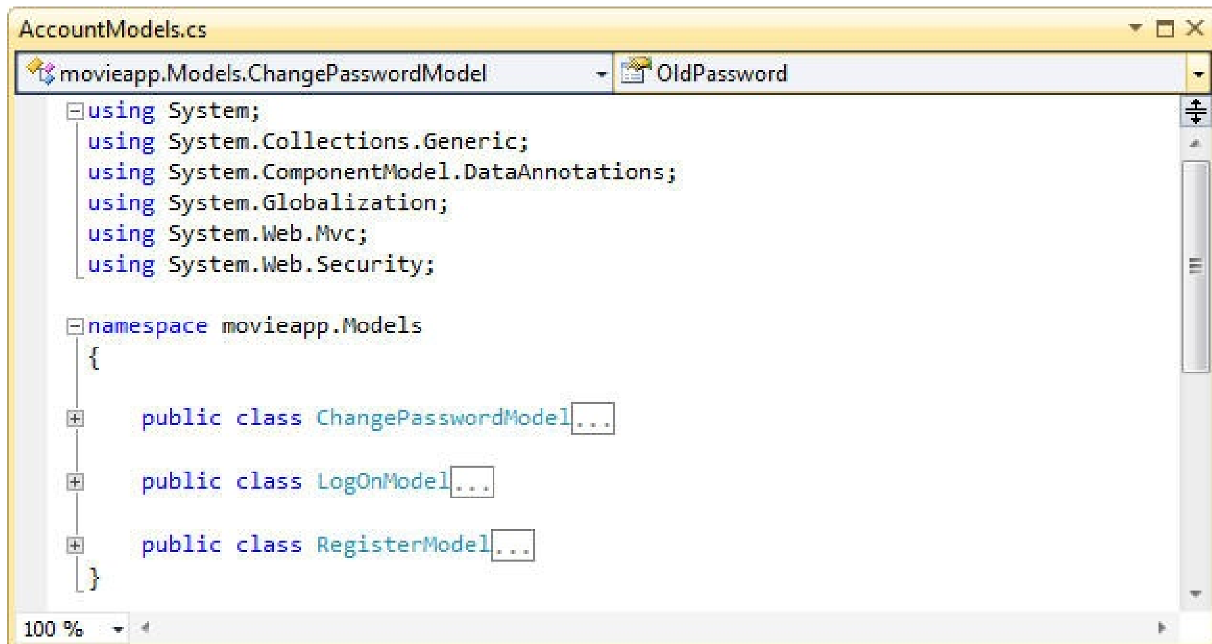
第 8 部分：添加安全。

MVC 应用程序安全

Models 文件夹包含表示应用程序模型的类。

Visual Web Developer 自动创建 **AccountModels.cs** 文件，该文件包含用于应用程序认证的模型。

AccountModels 包含 **LogOnModel**、**ChangePasswordModel** 和 **RegisterModel**：



Change Password 模型

```
public class ChangePasswordModel
{

[Required]
[DataType(DataType.Password)]
[Display(Name = "Current password")]
public string OldPassword { get; set; }

[Required]
[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", Minimu
[DataType(DataType.Password)]
[Display(Name = "New password")]
public string NewPassword { get; set; }

[DataType(DataType.Password)]
[Display(Name = "Confirm new password")]
[Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not
public string ConfirmPassword { get; set; }

}
```

Logon 模型

```
public class LogOnModel
{

[Required]
[Display(Name = "User name")]
public string UserName { get; set; }

[Required]
[DataType(DataType.Password)]
[Display(Name = "Password")]
public string Password { get; set; }

[Display(Name = "Remember me?")]
public bool RememberMe { get; set; }

}
```

Register 模型

```
public class RegisterModel
{

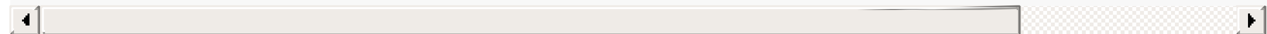
[Required]
[Display(Name = "User name")]
public string UserName { get; set; }

[Required]
[DataType(DataType.EmailAddress)]
[Display(Name = "Email address")]
public string Email { get; set; }

[Required]
[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", Minimu
[DataType(DataType.Password)]
[Display(Name = "Password")]
public string Password { get; set; }

[DataType(DataType.Password)]
[Display(Name = "Confirm password")]
[Compare("Password", ErrorMessage = "The password and confirmation password do not match.
public string ConfirmPassword { get; set; }

}
```



ASP.NET MVC - HTML 帮助器

HTML 帮助器用于修改 HTML 输出。

HTML 帮助器

通过 MVC，HTML 帮助器类似于传统的 ASP.NET Web Form 控件。

就像 ASP.NET 中的 Web Form 控件，HTML 帮助器用于修改 HTML。但是 HTML 帮助器是更轻量级的。与 Web Form 控件不同，HTML 帮助器没有事件模型和视图状态。

在大多数情况下，HTML 帮助器仅仅是一个返回字符串的方法。

通过 MVC，您可以创建您自己的帮助器，或者直接使用内建的 HTML 帮助器。

标准的 HTML 帮助器

MVC 包含了大多数常用的 HTML 元素类型的标准帮助器，比如 HTML 链接和 HTML 表单元素。

HTML 链接

呈现 HTML 链接的最简单的方法是使用 HTML.ActionLink() 帮助器。

通过 MVC，Html.ActionLink() 不连接到视图。它创建一个连接到控制器操作。

Razor 语法：

```
@Html.ActionLink("About this Website", "About")
```

ASP 语法：

```
<%=Html.ActionLink("About this Website", "About")%>
```

第一个参数是链接文本，第二个参数是控制器操作的名称。

上面的 Html.ActionLink() 帮助器，输出以下的 HTML：

```
<a href="/Home/About">About this Website</a>
```

Html.ActionLink() 帮助器的一些属性：

属性	描述
.linkText	URL 文本（标签），定位点元素的内部文本。
.actionName	操作（action）的名称。
.routeValues	传递给操作（action)的值，是一个包含路由参数的对象。
.controllerName	控制器的名称。
.htmlAttributes	URL 的属性设置，是一个包含要为该元素设置的 HTML 特性的对象。
.protocol	URL 协议，如 "http" 或 "https"。
.hostname	URL 的主机名。
.fragment	URL 片段名称（定位点名称）。

注释：您可以向控制器操作传递值。例如，您可以向数据库 Edit 操作传递数据库记录的 id：

Razor 语法 C#：

```
@Html.ActionLink("Edit Record", "Edit", new {Id=3})
```

Razor 语法 VB：

```
@Html.ActionLink("Edit Record", "Edit", New With{.Id=3})
```

上面的 Html.ActionLink() 帮助器，输出以下的 HTML：

```
<a href="/Home/Edit/3">Edit Record</a>
```

HTML 表单元素

以下 HTML 帮助器可用于呈现（修改和输出）HTML 表单元素：

- BeginForm()
- EndForm()
- TextArea()
- TextBox()
- CheckBox()
- RadioButton()
- ListBox()
- DropDownList()

- Hidden()
- Password()

ASP.NET 语法 C#：

```
<%= Html.ValidationSummary("Create was unsuccessful. Please correct the errors and try ag
<% using (Html.BeginForm()){%>
<p>
<label for="FirstName">First Name:</label>
<%= Html.TextBox("FirstName") %>
<%= Html.ValidationMessage("FirstName", "*") %>
</p>
<p>
<label for="LastName">Last Name:</label>
<%= Html.TextBox("LastName") %>
<%= Html.ValidationMessage("LastName", "*") %>
</p>
<p>
<label for="Password">Password:</label>
<%= Html.Password("Password") %>
<%= Html.ValidationMessage("Password", "*") %>
</p>
<p>
<label for="Password">Confirm Password:</label>
<%= Html.Password("ConfirmPassword") %>
<%= Html.ValidationMessage("ConfirmPassword", "*") %>
</p>
<p>
<label for="Profile">Profile:</label>
<%= Html.TextArea("Profile", new {cols=60, rows=10})%>
</p>
<p>
<%= Html.CheckBox("ReceiveNewsletter") %>
<label for="ReceiveNewsletter" style="display:inline">Receive Newsletter?</label>
</p>
<p>
<input type="submit" value="Register" />
</p>
<%}%>
```

ASP.NET MVC - 发布网站

学习如何在不使用 Visual Web Developer 的情况下发布 MVC 应用程序。

在不使用 Visual Web Developer 的情况下发布您的应用程序

通过在 WebMatrix、Visual Web Developer 或 Visual Studio 中使用发布命令，可以发布一个 ASP.NET MVC 应用程序到远程服务器上。

此功能会复制所有您的应用程序文件、控制器、模型、图像以及用于 MVC、Web Pages、Razor、Helpers、SQL Server Compact（如果使用数据库）所有必需的 DLL 文件。

有时您不希望使用这些选项。或许您的主机提供商仅支持 FTP？或许您的网站基于经典 ASP？或许您希望亲自拷贝这些文件？又或许您希望使用 Front Page、Expression Web 等其他一些发布软件？

您会遇到问题吗？是的，会的。但是您有办法解决它。

要执行网站复制，您必须知道如何引用正确的文件，哪些 DLL 文件需要复制，并在何处存储它们。

请按照下列步骤操作：

1. 使用最新版本的 ASP.NET

在您继续操作之前，请确保您的主机运行的是最新版的 ASP.NET（4.0 或者 4.5）。

2. 复制 Web 文件夹

从您的开发计算机上复制您的网站（所有文件夹和内容）到远程主机（服务器）上的应用程序文件夹中。

如果您的 **App_Data** 文件夹中包含测试数据，请不要复制这个 App_Data 文件夹（详见下面的第 5 点）。

3. 复制 DLL 文件

在远程服务器上的应用程序根目录中创建 bin 文件夹。（如果您已经安装 Helpers，则 bin 文件夹已经存在）

复制下列文件夹中的所有文件：

C:Program Files (x86)Microsoft ASP.NETASP.NET Web Pagesv1.0Assemblies

C:Program Files (x86)Microsoft ASP.NETASP.NET MVC 3Assemblies

到您的远程服务器上的应用程序的 bin 文件夹中。

4. 复制 SQL Server Compact DLL 文件

如果您的应用程序使用了 SQL Server Compact 数据库（在 App_Data 文件夹中的一个 .sdf 文件），那么您必须复制 SQL Server Compact DLL 文件：

复制下列文件夹中的所有文件：

C:Program Files (x86)Microsoft SQL Server Compact Editionv4.0Private

到您的远程服务器上的应用程序的 bin 文件夹中。

创建（或者编辑）应用程序的 Web.config 文件：

实例 C\

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<system.data>
<DbProviderFactories>
<remove invariant="System.Data.SqlServerCe.4.0" />

<add invariant="System.Data.SqlServerCe.4.0"
name="Microsoft SQL Server Compact 4.0"
description=".NET Framework Data Provider for Microsoft SQL Server Compact" type="System.

</DbProviderFactories>
</system.data>
</configuration>
```

5. 复制 SQL Server Compact 数据

您的 App_Data 文件夹中有没有包含测试数据的 .sdf 文件？

您是否希望发布您的测试数据到远程服务器上？

大多数时候一般是不希望。

如果您一定要复制 SQL 数据文件（.sdf 文件），那么您应该删除数据库中的所有数据，然后从您的开发计算机上复制一个空的 .sdf 文件到服务器上。

就是这样。**GOOD LUCK**！

Web Forms 教程

ASP.NET Web Forms - 教程

ASP.NET 是一个使用 HTML、CSS、JavaScript 和服务器脚本创建网页和网站的开发框架。

ASP.NET 支持三种不同的开发模式：
Web Pages（Web 页面）、MVC（Model View Controller 模型-视图-控制器）、Web Forms（Web 窗体）：

本教程介绍 **Web Forms**。

Web Pages
MVC
Web Forms

从何入手？

多数开发人员学习一个新技术，是从查看运行实例开始的。

如果您想查看一个 Web Forms 运行实例，请查看以下的 ASP.NET Web Forms 演示。

什么是 **Web Forms**？

Web Forms 是三种创建 ASP.NET 网站和 Web 应用程序的编程模式中的一种。

其他两种编程模式是 Web Pages 和 MVC（Model View Controller 模型-视图-控制器）。

Web Forms 是最古老的 ASP.NET 编程模式，是整合了 HTML、服务器控件和服务器代码的事件驱动网页。

Web Forms 是在服务器上编译和执行的，再由服务器生成 HTML 显示为网页。

Web Forms 有数以百计的 Web 控件和 Web 组件用来创建带有数据访问的用户驱动网站。

Visual Studio Express 2012/2010

Visual Studio Express 是 Microsoft Visual Studio 的免费版本。

Visual Studio Express 是为 Web Forms（和 MVC）量身定制的开发工具。

Visual Studio Express 包含：

- MVC 和 Web Forms

- 拖拽 Web 控件和 Web 组件
- Web 服务器语言（Razor 使用 VB 或者 C#）
- Web 服务器（IIS Express）
- 数据库服务器（SQL Server Compact）
- 完整的 Web 开发框架（ASP.NET）

如果您已经安装了 Visual Studio Express，您将从本教程中学到更多。

如果您想安装 Visual Studio Express，请点击下列链接中的一个：

Visual Web Developer 2012（Windows 7 或者 Windows 8）

Visual Web Developer 2010（Windows Vista 或者 XP）

ASP.NET 参考手册

在本教程的最后，您将看到一套完整的 ASP.NET 参考手册，介绍了对象、组件、属性和方法。

ASP.NET 参考手册

ASP.NET Web Forms - HTML 页面

简单的 ASP.NET 页面看上去就像普通的 HTML 页面。

Hello W3CSchool.cc

在开始学习 ASP.NET 之前，我们先来构建一个简单的 HTML 页面，该页面将在浏览器中显示 "Hello W3CSchool.cc"：

```
<table bgcolor="yellow" border="1" width="100%">

<tbody>

<tr><td>

## Hello W3CSchool.cc!

</td></tr>

</tbody>

</table>
```

用 HTML 编写的 Hello W3CSchool.cc

下面的代码将以 HTML 页面的形式显示实例：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3CSchool.cc!</h2>
</center>
</body>
</html>
```

如果您想亲自尝试一下，请保存上面的代码到一个名为 "**firstpage.htm**" 的文件中，并创建一个到该文件的链接：firstpage.htm。

用 ASP.NET 编写的 Hello W3CSchool.cc

转换 HTML 页面为 ASP.NET 页面最简单的方法是，直接复制一个 HTML 文件，并把新文件的扩展名改成 **.aspx** 。

下面的代码将以 ASP.NET 页面的形式显示实例：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3CSchool.cc!</h2>
</center>
</body>
</html>
```

如果您想亲自尝试一下，请保存上面的代码到一个名为 "**firstpage.aspx**" 的文件中，并创建一个到该文件的链接：firstpage.aspx。

它是如何工作的？

从根本上讲，ASP.NET 页面与 HTML 是完全相同的。

HTML 页面的扩展名是 .htm。如果浏览器向服务器请求一个 HTML 页面，服务器可以不进行任何修改，就直接发送页面给浏览器。

ASP.NET 页面的扩展名是 .aspx。如果浏览器向服务器请求个 ASP.NET 页面，服务器在将结果发回给浏览器之前，需要先处理页面中的可执行代码。

上面的 ASP.NET 页面不包含任何可执行的代码，所以没有执行任何东西。在下面的实例中，我们将添加一些可执行的代码到页面中，以便演示静态 HTML 页面和动态 ASP 页面的不同之处。

经典 ASP

Active Server Pages (ASP) 已经流行很多年了。通过 ASP，可以在 HTML 页面中放置可执行代码。

之前的 ASP 版本（在 ASP.NET 之前）通常被称为经典 ASP。

ASP.NET 不完全兼容经典 ASP，但是只需要经过少量的修改，大部分经典 ASP 页面就可以作为 ASP.NET 页面良好地运行。

如果您想学习更多关于经典 ASP 的知识，请访问我们的 ASP 教程。

用经典 ASP 编写的动态页面

为了演示 ASP 是如何显示包含动态内容的页面，我们将向上面的实例中添加一些可执行的代码（红色字体标识）：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3CSchool.cc!</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

<% --%> 标签内的代码是在服务器上执行的。

Response.Write 是用来向 HTML 输出流中写东西的 ASP 代码。

Now() 是一个返回服务器当前日期和时间的函数。

如果您想亲自尝试一下，请保存上面的代码到一个名为 "**dynpage.asp**" 的文件中，并创建一个到该文件的链接：dynpage.asp。

用 ASP .NET 编写的动态页面

下面的代码将以 ASP.NET 页面的形式显示实例：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3CSchool.cc!</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

如果您想亲自尝试一下，请保存上面的代码到一个名为 "**dynpage.aspx**" 的文件中，并创建一个到该文件的链接：dynpage.aspx。

ASP.NET 对比经典 ASP

上面的实例无法演示 ASP.NET 与经典 ASP 之间任何的不同之处。

正如最后的两个实例中，您看不出 ASP 页面和 ASP.NET 页面两者之间的不同之处。

在下一章中，您将看到服务器控件是如何让 ASP.NET 比经典 ASP 更强大的。

ASP.NET Web Forms - 服务器控件

服务器控件是服务器可理解的标签。

经典 ASP 的局限性

下面列出的代码是从上一章中复制的：

```
<html>
<body bgcolor="yellow">
<center>
<h2>Hello W3CSchool.cc!</h2>
<p><%Response.Write(now())%></p>
</center>
</body>
</html>
```

上面的代码反映出经典 ASP 的局限性：代码块必须放置在您想要输出显示的位置。

通过经典 ASP，想要把可执行代码从 HTML 页面中分离出来是不可能的。这让页面变得难以阅读，也难以维护。

ASP.NET - 服务器控件

ASP.NET 通过服务器控件，已经解决了上述的"意大利面条式代码"问题。

服务器控件是服务器可理解的标签。

有三种类型的服务器控件：

- HTML 服务器控件 - 创痛的 HTML 标签
- Web 服务器控件 - 新的 ASP.NET 标签
- Validation 服务器控件 - 用于输入验证

ASP.NET - HTML 服务器控件

HTML 服务器控件是服务器可理解的 HTML 标签。

ASP.NET 文件中的 HTML 元素，默认是作为文本进行处理的。要想让这些元素可编程，需向 HTML 元素中添加 runat="server" 属性。这个属性表示，该元素将被作为服务器控件进行处理。同时需要添加 id 属性来标识服务器控件。id 引用可用于操作运行时的服务器控件。

注释：所有 HTML 服务器控件必须位于带有 runat="server" 属性的 <form> 标签内。runat="server" 属性表明了该表单必须在服务器上进行处理。同时也表明了包含在它内部的控件可被服务器脚本访问。

在下面的实例中，我们在 .aspx 文件中声明了一个 HtmlAnchor 服务器控件。然后我们在一个事件句柄（事件句柄是一种针对给定事件执行代码的子例程）中操作 HtmlAnchor 控件的 HRef 属性。Page_Load 事件是 ASP.NET 可理解的多种事件中的一种：

```
<script runat="server">
Sub Page_Load
link1.HRef="http://www.w3cschool.cc"
End Sub
</script>

<html>
<body>

<form runat="server">
<a id="link1" runat="server">Visit W3CSchool.cc!</a>
</form>

</body>
</html>
```

可执行代码本身已经被移到 HTML 之外了。

ASP.NET - Web 服务器控件

Web 服务器控件是服务器可理解的特殊 ASP.NET 标签。

就像 HTML 服务器控件，Web 服务器控件也是在服务器上创建的，它们同样需要 runat="server" 属性才能生效。然而，Web 服务器控件没有必要映射任何已存在的 HTML 元素，它们可以表示更复杂的元素。

创建 Web 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

在下面的实例中，我们在 .aspx 文件中声明了一个 Button 服务器控件。然后我们为 Click 事件创建一个事件句柄，用来改变按钮上的文本：

```
<script runat="server">
Sub submit(Source As Object, e As EventArgs)
button1.Text="You clicked me!"
End Sub
</script>

<html>
<body>

<form runat="server">
<asp:Button id="button1" Text="Click me!"
runat="server" OnClick="submit"/>
</form>

</body>
</html>
```

ASP.NET - Validation 服务器控件

Validation 服务器控件是用来验证用户输入的。如果用户输入没有通过验证，将显示一条错误消息给用户。

每种 validation 控件执行一种指定类型的验证（比如验证摸个指定的值或者某个范围的值）。

在默认情况下，当 Button、ImageButton、LinkButton 控件被点击时，会执行页面验证。您可以设置 CausesValidation 为 false， 来阻止按钮控件被点击时进行验证。

创建 Validation 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

在下面的实例中，我们在 .aspx 文件中声明了一个 TextBox 控件、一个 Button 控件、一个 RangeValidator 控件。如果验证失败，文本 "The value must be from 1 to 100!" 将会显示在 RangeValidator 控件中：

实例

```
<html>
<body>

<form runat="server">
<p>Enter a number from 1 to 100:
<asp:TextBox id="tbox1" runat="server" />
<br /><br />
<asp:Button Text="Submit" runat="server" />
</p>

<p>
<asp:RangeValidator
ControlToValidate="tbox1"
MinimumValue="1"
MaximumValue="100"
Type="Integer"
Text="The value must be from 1 to 100!"
runat="server" />
</p>
</form>

</body>
</html>
```

ASP.NET Web Forms - 事件

事件句柄是一种针对给定事件来执行代码的子例程。

ASP.NET - 事件句柄

请看下面的代码：

```
<%
lbl1.Text="The date and time is " & now()
%>

<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
</form>
</body>
</html>
```

上面的代码将在何时被执行？答案是："不知道..."。

Page_Load 事件

Page_Load 事件是 ASP.NET 可理解的众多事件之一。Page_Load 事件会在页面加载时被触发， ASP.NET 将自动调用 Page_Load 子例程，并执行其中的代码：

实例

```
<script runat="server">
Sub Page_Load
lbl1.Text="The date and time is " & now()
End Sub
</script>

<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
</form>
</body>
</html>
```

演示实例？

注释：Page_Load 事件不包含对象引用或事件参数！

Page.IsPostBack 属性

Page_Load 子例程会在页面每次加载时运行。如果您只想在页面第一次加载时执行 Page_Load 子例程中的代码，那么您可以使用 Page.IsPostBack 属性。如果 Page.IsPostBack 属性设置为 false，则页面第一次被载入，如果设置为 true，则页面被传回到服务器（比如，通过点击表单上的按钮）：

实例

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
lbl1.Text="The date and time is " & now()
end if
End Sub

Sub submit(s As Object, e As EventArgs)
lbl2.Text="Hello World!"
End Sub
</script>

<html>
<body>
<form runat="server">
<h3><asp:label id="lbl1" runat="server" /></h3>
<h3><asp:label id="lbl2" runat="server" /></h3>
<asp:button text="Submit" onclick="submit" runat="server" />
</form>
</body>
</html>
```

演示实例？

上面的实例仅在页面第一次加载时显示 "The date and time is...." 消息。当用户点击 Submit 按钮是，submit 子例程将会在第二个 label 中写入 "Hello World!"，但是第一个 label 中的日期和时间不会改变。

ASP.NET Web Forms - HTML 表单

所有的服务器控件都必须出现在 <form> 标签中，<form> 标签必须包含 runat="server" 属性。

ASP.NET Web 表单

所有的服务器控件都必须出现在 <form> 标签中，<form> 标签必须包含 runat="server" 属性。runat="server" 属性表明该表单必须在服务器上进行处理。同时也表明了包含在它内部的控件可被服务器脚本访问：

```
<form runat="server">
...HTML + server controls
</form>
```

注释：该表单总是被提交到自身页面。如果您指定了一个 action 属性，它会被忽略。如果您省略了 metion 属性，它将会默认设置 method="post"。同时，如果您没有指定 name 和 id 属性，它们会由 ASP.NET 自动分配。

注释：一个 .aspx 页面只能包含一个 <form runat="server"> 控件！

如果您在一个包含不带有 name、method、action 或 id 属性的表单的 .aspx 页面中选择查看源代码，您会看到 ASP.NET 添加这些属性到表单上了，如下所示：

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
...some code
</form>
```

提交表单

表单通常通过点击按钮来提交。ASP.NET 中的 Button 服务器控件的格式如下：

```
<asp:Button id="id" text="label" OnClick="sub" runat="server" />
```

id 属性为按钮定义了一个唯一的名称，text 属性为按钮分配了一个标签。onClick 事件句柄规定了一个要执行的已命名的子例程。

在下面的实例中，我们在 .aspx 文件中声明了一个 Button 控件。点击按钮运行改变按钮上文本的子例程：

实例

ASP.NET Web Forms - 维持 ViewState

通过在您的 Web Form 中维持对象的 ViewState（视图状态），您可以省去大量的编码工作。

维持 ViewState（视图状态）

在经典 ASP 中，当一个表单被提交时，所有的表单值都会被清空。假设您提交了一个带有大量信息的表单，而服务器返回了一个错误。您不得不回到表单改正信息。您点击返回按钮，然后发生了什么......所有表单值都被清空了，您不得不重新开始所有的一切！站点没有维持您的 ViewState。

在 ASP .NET 中，当一个表单被提交时，表单会连同表单值一起出现在浏览器窗口中。如何做到的呢？这是因为 ASP .NET 维持了您的 ViewState。 ViewState 会在页面被提交到服务器时表明它的状态。这个状态是通过在带有 <form runat="server"> 控件的每个页面上放置一个隐藏域定义的。源代码如下所示：

```
<form name="_ctl0" method="post" action="page.aspx" id="_ctl0">
<input type="hidden" name="__VIEWSTATE"
value="dDwtNTI0ODU5MDE1Ozs+ZBCF2ryjMpeVgUrY2eTj79HNl4Q=" />

.....some code

</form>
```

维持 ViewState 是 ASP.NET Web Forms 的默认设置。如果您想不维持 ViewState，请在 .aspx 页面顶部包含指令 <%@ Page EnableViewState="false" %>，或者向任意控件添加属性 EnableViewState="false" 。

请看下面的 .aspx 文件。它演示了"老"的运行方式。当您点击提交按钮，表单值将会消失：

实例

```
<html>
<body>

<form action="demo_classicasp.aspx" method="post">
Your name: <input type="text" name="fname" size="20">
<input type="submit" value="Submit">
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>"" Then
Response.Write("Hello " & fname & "!")
End If
%>

</body>
</html>
```

演示实例？

下面是新的 ASP .NET 方式。当您点击提交按钮，表单值不会消失：

实例

点击实例的右边框架中的查看源代码，您将看到 ASP .NET 已经在表单中添加了一个隐藏域来维持 ViewState。

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lbl1.Text="Hello " & txt1.Text & "!"
End Sub
</script>

<html>
<body>

<form runat="server">
Your name: <asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例？

ASP.NET Web Forms - TextBox 控件

TextBox 控件用于创建用户可输入文本的文本框。

TextBox 控件

TextBox 控件用于创建用户可输入文本的文本框。

TextBox 控件的特性和属性列在我们的 WebForms 控件参考手册页面。

下面的实例演示了您可能会用到的 TextBox 控件的一些属性：

实例

```
<html>
<body>

<form runat="server">

A basic TextBox:
<asp:TextBox id="tb1" runat="server" />
<br /><br />

A password TextBox:
<asp:TextBox id="tb2" TextMode="password" runat="server" />
<br /><br />

A TextBox with text:
<asp:TextBox id="tb4" Text="Hello World!" runat="server" />
<br /><br />

A multiline TextBox:
<asp:TextBox id="tb3" TextMode="multiline" runat="server" />
<br /><br />

A TextBox with height:
<asp:TextBox id="tb6" rows="5" TextMode="multiline"
runat="server" />
<br /><br />

A TextBox with width:
<asp:TextBox id="tb5" columns="30" runat="server" />

</form>

</body>
</html>
```

演示实例？

添加脚本

当表单被提交时，TextBox 控件的内容和设置可能会被服务器脚本修改。表单可通过点击一个按钮或当用户修改 TextBox 控件的值的时候进行提交。

在下面的实例中，我们在 .aspx 文件中声明了一个 TextBox 控件、一个 Button 控件和一个 Label 控件。当提交按钮被触发时，submit 子例程将被执行。submit 子例程将写入一行文本到 Label 控件中：

实例

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lbl1.Text="Your name is " & txt1.Text
End Sub
</script>

<html>
<body>

<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例？

在下面的实例中，我们在 .aspx 文件中声明了一个 TextBox 控件和一个 Label 控件。当您修改了 TextBox 中的值，并且在 TextBox 外部点击（或者按下了 Tab 键）时，change 子例程将会被执行。change 子例程将写入一行文本到 Label 控件中：

实例

```
<script runat="server">
Sub change(sender As Object, e As EventArgs)
lbl1.Text="You changed text to " & txt1.Text
End Sub
</script>

<html>
<body>

<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server"
text="Hello World!"
ontextchanged="change" autopostback="true"/>
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例？

ASP.NET Web Forms - Button 控件

Button 控件用于显示一个下压按钮。

Button 控件

Button 控件用于显示一个下压按钮。下压按钮可能是一个提交按钮或者是一个命令按钮。在默认情况下，这个控件是提交按钮。

提交按钮没有命令名称，当它被点击时，它会把页面传回到服务器。您可以编写一些事件句柄，当提交按钮被点击时，用来控制动作的执行。

命令按钮有命令名称，并且允许您在页面上创建多个 Button 控件。您可以编写一些时间句柄，当命令按钮被点击时，用来控制动作的执行。

Button 控件的特性和属性列在我们的 WebForms 控件参考手册页面。

下面的实例演示了一个简单的 Button 控件：

```
<html>
<body>

<form runat="server">
<asp:Button id="b1" Text="Submit" runat="server" />
</form>

</body>
</html>
```

添加脚本

表单通常通过点击按钮进行提交。

在下面的实例中，我们在 .aspx 文件中声明了一个 TextBox 控件、一个 Button 控件和一个 Label 控件。当提交按钮被触发时，submit 子例程将被执行。submit 子例程将写入一行文本到 Label 控件中：

实例

```
<script runat="server">
Sub submit(sender As Object, e As EventArgs)
lbl1.Text="Your name is " & txt1.Text
End Sub
</script>

<html>
<body>

<form runat="server">
Enter your name:
<asp:TextBox id="txt1" runat="server" />
<asp:Button OnClick="submit" Text="Submit" runat="server" />
<p><asp:Label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例？

ASP.NET Web Forms - 数据绑定

我们可以使用数据绑定（Data Binding）来完成带可选项的列表，这些可选项来自某个导入的数据源，比如数据库、XML 文件或者脚本。

数据绑定

下面的控件是支持数据绑定的列表控件：

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

以上每个控件的可选项通常是在一个或者多个 asp:ListItem 控件中定义，如下：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="countrylist" runat="server">
<asp:ListItem value="N" text="Norway" />
<asp:ListItem value="S" text="Sweden" />
<asp:ListItem value="F" text="France" />
<asp:ListItem value="I" text="Italy" />
</asp:RadioButtonList>
</form>

</body>
</html>
```

然而，我们可以使用某种独立的数据源进行数据绑定，比如数据库、XML 文件或者脚本，通过数据绑定来填充列表的可选项。

通过使用导入的数据源，数据从 HTML 中分离出来，并且对可选项的修改都是在独立的数据源中完成的。

在下面的三个章节中，我们将描述如何从脚本化的数据源中绑定数据。

ASP.NET Web Forms - ArrayList 对象

ArrayList 对象是包含单个数据值的项目的集合。



尝试一下 - 实例

ArrayList DropDownList

ArrayList RadioButtonList

创建 ArrayList

ArrayList 对象是包含单个数据值的项目的集合。

通过 Add() 方法向 ArrayList 添加项目。

下面的代码创建了一个名为 mycountries 的 ArrayList 对象，并添加了四个项目：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
end if
end sub
</script>
```

在默认情况下，一个 ArrayList 对象包含 16 个条目。可通过 TrimToSize() 方法把 ArrayList 调整为最终尺寸：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
end if
end sub
</script>
```

通过 Sort() 方法，ArrayList 也能够按照字母顺序或者数字顺序进行排序：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
mycountries.Sort()
end if
end sub
</script>
```

要实现反向排序，请在 Sort() 方法后应用 Reverse() 方法：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
mycountries.Sort()
mycountries.Reverse()
end if
end sub
</script>
```

绑定数据到 ArrayList

ArrayList 对象可为下列的控件自动生成文本和值：

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

为了绑定数据到 RadioButtonList 控件，首先要在 .aspx 页面中创建一个 RadioButtonList 控件（不带任何 asp:ListItem 元素）：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>

</body>
</html>
```

然后添加创建列表的脚本，并且绑定列表中的值到 RadioButtonList 控件：

实例

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New ArrayList
mycountries.Add("Norway")
mycountries.Add("Sweden")
mycountries.Add("France")
mycountries.Add("Italy")
mycountries.TrimToSize()
mycountries.Sort()
rb.DataSource=mycountries
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>

</body>
</html>
```

演示实例？

RadioButtonList 控件的 DataSource 属性被设置为该 ArrayList，它定义了这个 RadioButtonList 控件的数据源。RadioButtonList 控件的 DataBind() 方法把 RadioButtonList 控件与数据源绑定在一起。

注释：数据值作为控件的 Text 和 Value 属性来使用。如需添加不同于 Text 的 Value，请使用 Hashtable 对象或者 SortedList 对象。

ASP.NET Web Forms - Hashtable 对象

Hashtable 对象包含用键/值对表示的项目。

尝试一下 - 实例

Hashtable RadiobuttonList 1

Hashtable RadiobuttonList 2

Hashtable DropDownList

创建 Hashtable

Hashtable 对象包含用键/值对表示的项目。键被用作索引，通过搜索键，可以实现对值的快速搜索。

通过 Add() 方法向 Hashtable 添加项目。

下面的代码创建了一个名为 mycountries 的 Hashtable 对象，并添加了四个元素：

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New Hashtable
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
end if
end sub
</script>
```

数据绑定

Hashtable 对象可为下列的控件自动生成文本和值：

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

为了绑定数据到 RadioButtonList 控件，首先要在 .aspx 页面中创建一个 RadioButtonList 控件（不带任何 asp:ListItem 元素）：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后添加创建列表的脚本，并且绑定列表中的值到 RadioButtonList 控件：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New Hashtable
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后我们添加一个子例程，当用户点击 RadioButtonList 控件中的某个项目时，该子例程会被执行。当某个单选按钮被点击时，label 中会出现一行文本：

实例

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New Hashtable
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub

sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例？

注释：您无法选择添加到 Hashtable 的项目的排序方式。如需对项目进行字母排序或者数字排序，请使用 SortedList 对象。

ASP.NET Web Forms - SortedList 对象

SortedList 对象结合了 ArrayList 对象和 Hashtable 对象的特性。

尝试一下 - 实例

SortedList RadiobuttonList 1

SortedList RadiobuttonList 2

SortedList DropDownList

SortedList 对象

SortedList 对象包含用键/值对表示的项目。SortedList 对象按照字母顺序或者数字顺序自动地对项目进行排序。

通过 Add() 方法向 SortedList 添加项目。通过 TrimToSize() 方法把 SortedList 调整为最终尺寸。

下面的代码创建了一个名为 mycountries 的 SortedList 对象，并添加了四个元素：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New SortedList
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
end if
end sub
</script>
```

数据绑定

SortedList 对象可为下列的控件自动生成文本和值：

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

为了绑定数据到 RadioButtonList 控件，首先要在 .aspx 页面中创建一个 RadioButtonList 控件（不带任何 asp:ListItem 元素）：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后添加创建列表的脚本，并且绑定列表中的值到 RadioButtonList 控件：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New SortedList
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后我们添加一个子例程，当用户点击 RadioButtonList 控件中的某个项目时，该子例程会被执行。当某个单选按钮被点击时，label 中会出现一行文本：

实例

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New SortedList
mycountries.Add("N","Norway")
mycountries.Add("S","Sweden")
mycountries.Add("F","France")
mycountries.Add("I","Italy")
rb.DataSource=mycountries
rb.DataValueField="Key"
rb.DataTextField="Value"
rb.DataBind()
end if
end sub

sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例？

ASP.NET Web Forms - XML 文件

我们可以绑定 XML 文件到列表控件。

一个 XML 文件

这里有一个名为 "countries.xml" 的 XML 文件：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<countries>

<country>
<text>Norway</text>
<value>N</value>
</country>

<country>
<text>Sweden</text>
<value>S</value>
</country>

<country>
<text>France</text>
<value>F</value>
</country>

<country>
<text>Italy</text>
<value>I</value>
</country>

</countries>
```

查看这个 XML 文件：countries.xml

绑定 DataSet 到 List 控件

首先，导入 "System.Data" 命名空间。我们需要该命名空间与 DataSet 对象一起工作。把下面这条指令包含在 .aspx 页面的顶部：

```
<%@ Import Namespace="System.Data" %>
```

接着，为 XML 文件创建一个 DataSet，并在页面第一次加载时把这个 XML 文件载入 DataSet：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New DataSet
mycountries.ReadXml(MapPath("countries.xml"))
end if
end sub
```

为了绑定数据到 RadioButtonList 控件，首先要在 .aspx 页面中创建一个 RadioButtonList 控件（不带任何 asp:ListItem 元素）：

```
<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>

</body>
</html>
```

然后添加创建 XML DataSet 的脚本，并且绑定 XML DataSet 中的值到 RadioButtonList 控件：

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New DataSet
mycountries.ReadXml(MapPath("countries.xml"))
rb.DataSource=mycountries
rb.DataValueField="value"
rb.DataTextField="text"
rb.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
</form>

</body>
</html>
```

然后我们添加一个子例程，当用户点击 RadioButtonList 控件中的某个项目时，该子例程会被执行。当某个单选按钮被点击时，label 中会出现一行文本：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycountries=New DataSet
mycountries.ReadXml(MapPath("countries.xml"))
rb.DataSource=mycountries
rb.DataValueField="value"
rb.DataTextField="text"
rb.DataBind()
end if
end sub

sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>

<html>
<body>

<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>

</body>
</html>
```

演示实例？

ASP.NET Web Forms - Repeater 控件

Repeater 控件用于显示被绑定在该控件上的项目的重复列表。

绑定 DataSet 到 Repeater 控件

Repeater 控件用于显示被绑定在该控件上的项目的重复列表。Repeater 控件可被绑定到数据库表、XML 文件或者其他项目列表。在这里，我们将演示如何绑定 XML 文件到 Repeater 控件。

在我们的实例中，我们将使用下面的 XML 文件（"cdcatalog.xml"）：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
</cd>
<cd>
<title>Greatest Hits</title>
<artist>Dolly Parton</artist>
<country>USA</country>
<company>RCA</company>
<price>9.90</price>
<year>1982</year>
</cd>
<cd>
<title>Still got the blues</title>
<artist>Gary Moore</artist>
<country>UK</country>
<company>Virgin records</company>
<price>10.20</price>
<year>1990</year>
</cd>
<cd>
<title>Eros</title>
<artist>Eros Ramazzotti</artist>
<country>EU</country>
<company>BMG</company>
<price>9.90</price>
<year>1997</year>
</cd>
</catalog>
```

查看这个 XML 文件：cdcatalog.xml

首先，导入 "System.Data" 命名空间。我们需要该命名空间与 DataSet 对象一起工作。 把下面这条指令包含在 .aspx 页面的顶部：

```
<%@ Import Namespace="System.Data" %>
```

接着，为 XML 文件创建一个 DataSet，并在页面第一次加载时把这个 XML 文件载入 DataSet：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
end if
end sub
```

然后我们在 .aspx 页面中创建一个 Repeater 控件。<HeaderTemplate> 元素中的内容被首先呈现，并且在输出中仅出现一次，而 <ItemTemplate> 元素中的内容会对应 DataSet 中的每条 "record" 重复出现，最后，<FooterTemplate> 元素中的内容在输出中仅出现一次：

```
<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">

<HeaderTemplate>
...
</HeaderTemplate>

<ItemTemplate>
...
</ItemTemplate>

<FooterTemplate>
...
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

然后我们添加创建 DataSet 的脚本，并且绑定 mycdcatalog DataSet 到 Repeater 控件。然后使用 HTML 标签来填充 Repeater 控件，并通过 <%#Container.DataItem("fieldname")%> 绑定数据项目到 <ItemTemplate> 区域内的单元格中：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">

<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

演示实例？

使用 <AlternatingItemTemplate>

您可以在 <ItemTemplate> 元素后添加 <AlternatingItemTemplate> 元素，用来描述输出中交替行的外观。在下面的实例中，表格每隔一行就会显示为浅灰色的背景：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">

<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>

<AlternatingItemTemplate>
<tr bgcolor="#e8e8e8">
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</AlternatingItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

演示实例？

使用 <SeparatorTemplate>

<SeparatorTemplate> 元素用于描述每个记录之间的分隔符。在下面的实例中，每个表格行之间插入了一条水平线：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">

<HeaderTemplate>
<table border="0" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>

<SeparatorTemplate>
<tr>
<td colspan="6"><hr /></td>
</tr>
</SeparatorTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

演示实例 ?

ASP.NET Web Forms - DataList 控件

DataList 控件，类似于 Repeater 控件，用于显示绑定在该控件上的项目的重复列表。不过，DataList 控件会默认地在数据项目上添加表格。

绑定 DataSet 到 DataList 控件

DataList 控件，类似于 Repeater 控件，用于显示绑定在该控件上的项目的重复列表。不过，DataList 控件会默认地在数据项目上添加表格。DataList 控件可被绑定到数据库表、XML 文件或者其他项目列表。在这里，我们将演示如何绑定 XML 文件到 DataList 控件。

在我们的实例中，我们将使用下面的 XML 文件（"cdcatalog.xml"）：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<catalog>
<cd>
<title>Empire Burlesque</title>
<artist>Bob Dylan</artist>
<country>USA</country>
<company>Columbia</company>
<price>10.90</price>
<year>1985</year>
</cd>
<cd>
<title>Hide your heart</title>
<artist>Bonnie Tyler</artist>
<country>UK</country>
<company>CBS Records</company>
<price>9.90</price>
<year>1988</year>
</cd>
<cd>
<title>Greatest Hits</title>
<artist>Dolly Parton</artist>
<country>USA</country>
<company>RCA</company>
<price>9.90</price>
<year>1982</year>
</cd>
<cd>
<title>Still got the blues</title>
<artist>Gary Moore</artist>
<country>UK</country>
<company>Virgin records</company>
<price>10.20</price>
<year>1990</year>
</cd>
<cd>
<title>Eros</title>
<artist>Eros Ramazzotti</artist>
<country>EU</country>
<company>BMG</company>
<price>9.90</price>
<year>1997</year>
</cd>
</catalog>
```

查看这个 XML 文件：cdcatalog.xml

首先，导入 "System.Data" 命名空间。我们需要该命名空间与 DataSet 对象一起工作。 把下面这条指令包含在 .aspx 页面的顶部：

```
<%@ Import Namespace="System.Data" %>
```

接着，为 XML 文件创建一个 DataSet，并在页面第一次加载时把这个 XML 文件载入 DataSet：

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
end if
end sub
```

然后我们在 .aspx 页面中创建一个 DataList 控件。<HeaderTemplate> 元素中的内容被首先呈现，并且在输出中仅出现一次，而 <ItemTemplate> 元素中的内容会对应 DataSet 中的每条 "record" 重复出现，最后，<FooterTemplate> 元素中的内容在输出中仅出现一次：

```
<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog" runat="server">

<HeaderTemplate>
...
</HeaderTemplate>

<ItemTemplate>
...
</ItemTemplate>

<FooterTemplate>
...
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>
```

然后我们添加创建 DataSet 的脚本，并且绑定 mycdcatalog DataSet 到 DataList 控件。然后使用包含表头的 <HeaderTemplate>、包含要显示的数据项的 <ItemTemplate> 和包含文本的 <FooterTemplate> 来填充 DataList 控件。请注意，可设置 DataList 的 gridlines 属性为 "both" 来显示表格边框：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog"
gridlines="both" runat="server">

<HeaderTemplate>
My CD Catalog
</HeaderTemplate>

<ItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</ItemTemplate>

<FooterTemplate>
Copyright Hege Refsnes
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>
```

演示实例 ?

使用样式

您也可以向 DataList 控件添加样式，让输出更加花哨：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog"
runat="server"
cellpadding="2"
cellspacing="2"
borderstyle="inset"
backcolor="#e8e8e8"
width="100%"
headerstyle-font-name="Verdana"
headerstyle-font-size="12pt"
headerstyle-horizontalalign="center"
headerstyle-font-bold="true"
itemstyle-backcolor="#778899"
itemstyle-forecolor="#ffffff"
footerstyle-font-size="9pt"
footerstyle-font-italic="true">

<HeaderTemplate>
My CD Catalog
</HeaderTemplate>

<ItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</ItemTemplate>

<FooterTemplate>
Copyright Hege Refsnes
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>
```

演示实例 ?

使用 <AlternatingItemTemplate>

您可以在 <ItemTemplate> 元素后添加 <AlternatingItemTemplate> 元素，用来描述输出中交替行的外观。您可以在 DataList 控件内部对 <AlternatingItemTemplate> 区域的数据添加样式：

实例

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
dim mycdcatalog=New DataSet
mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
cdcatalog.DataSource=mycdcatalog
cdcatalog.DataBind()
end if
end sub
</script>

<html>
<body>

<form runat="server">
<asp:DataList id="cdcatalog"
runat="server"
cellpadding="2"
cellspacing="2"
borderstyle="inset"
backcolor="#e8e8e8"
width="100%"
headerstyle-font-name="Verdana"
headerstyle-font-size="12pt"
headerstyle-horizontalalign="center"
headerstyle-font-bold="True"
itemstyle-backcolor="#778899"
itemstyle-forecolor="#ffffff"
alternatingitemstyle-backcolor="#e8e8e8"
alternatingitemstyle-forecolor="#000000"
footerstyle-font-size="9pt"
footerstyle-font-italic="True">

<HeaderTemplate>
My CD Catalog
</HeaderTemplate>

<ItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</ItemTemplate>

<AlternatingItemTemplate>
"<%#Container.DataItem("title")%>" of
<%#Container.DataItem("artist")%> -
$<%#Container.DataItem("price")%>
</AlternatingItemTemplate>

<FooterTemplate>
&copy; Hege Refsnes
</FooterTemplate>

</asp:DataList>
</form>

</body>
</html>
```

演示实例 ?

ASP.NET Web Forms - 数据库连接

ADO.NET 也是 .NET 框架的组成部分。ADO.NET 用于处理数据访问。通过 ADO.NET，您可以操作数据库。



尝试一下 - 实例

数据库连接 - 绑定到 DataList 控件

数据库连接 - 绑定到 Repeater 控件

什么是 ADO.NET？

- ADO.NET 是 .NET 框架的组成部分
- ADO.NET 由一系列用于处理数据访问的类组成
- ADO.NET 完全基于 XML
- ADO.NET 没有 Recordset 对象，这一点与 ADO 不同

创建数据库连接

在我们的实例中，我们将使用 Northwind 数据库。

首先，导入 "System.Data.OleDb" 命名空间。我们需要这个命名空间来操作 Microsoft Access 和其他 OLE DB 数据库提供商。我们将在 Page_Load 子例程中创建这个数据库的连接。我们创建一个 dbconn 变量，并为其赋值一个新的 OleDbConnection 类，这个类带有指示 OLE DB 提供商和数据库位置的连接字符串。然后我们打开数据库连接：

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
end sub
</script>
```

注释：这个连接字符串必须是没有折行的连续字符串！

创建数据库命令

为了指定需从数据库取回的记录，我们将创建一个 dbcomm 变量，并为其赋值一个新的 OleDbCommand 类。这个 OleDbCommand 类用于发出针对数据库表的 SQL 查询：

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
end sub
</script>
```

创建 DataReader

OleDbDataReader 类用于从数据源中读取记录流。DataReader 是通过调用 OleDbCommand 对象的 ExecuteReader 方法来创建的：

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
end sub
</script>
```

绑定到 Repeater 控件

然后，我们绑定 DataReader 到 Repeater 控件：

实例

```
<%@ Import Namespace="System.Data.OleDb" %>

<script runat="server">
sub Page_Load
dim dbconn,sql,dbcomm,dbread
dbconn=New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
data source=" & server.mappath("northwind.mdb"))
dbconn.Open()
sql="SELECT * FROM customers"
dbcomm=New OleDbCommand(sql,dbconn)
dbread=dbcomm.ExecuteReader()
customers.DataSource=dbread
customers.DataBind()
dbread.Close()
dbconn.Close()
end sub
</script>

<html>
<body>

<form runat="server">
<asp:Repeater id="customers" runat="server">

<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Companyname</th>
<th>Contactname</th>
<th>Address</th>
<th>City</th>
</tr>
</HeaderTemplate>

<ItemTemplate>
<tr>
<td><%#Container.DataItem("companyname")%></td>
<td><%#Container.DataItem("contactname")%></td>
<td><%#Container.DataItem("address")%></td>
<td><%#Container.DataItem("city")%></td>
</tr>
</ItemTemplate>

<FooterTemplate>
</table>
</FooterTemplate>

</asp:Repeater>
</form>

</body>
</html>
```

演示实例？

关闭数据库连接

如果不再需要访问数据库，请记得关闭 DataReader 和数据库连接：

```
dbread.Close()
dbconn.Close()
```

ASP.NET Web Forms - 母版页

母版页为您的网站的其他页面提供模版。

母版页

母版页允许您为您的 web 应用程序中的所有页面（或页面组）创建一致的外观和行为。

母版页为其他页面提供模版，带有共享的布局和功能。母版页为内容定义了可被内容页覆盖的占位符。输出结果是母版页和内容页的组合。

内容页包含您想要显示的内容。

当用户请求内容页时，ASP.NET 会对页面进行合并以生成结合了母版页布局和内容页内容的输出。

母版页实例

```
<%@ Master %>

<html>
<body>
<h1>Standard Header From Masterpage</h1>
<asp:ContentPlaceHolder id="CPH1" runat="server">
</asp:ContentPlaceHolder>
</body>
</html>
```

上面的母版页是一个为其他页面设计的普通 HTML 模版页。

@ Master 指令定义它为一个母版页。

母版页为单独的内容包含占位标签 **<asp:ContentPlaceHolder>**。

id="CPH1" 属性标识占位符，在相同母版页中允许多个占位符。

这个母版页被保存为 **"master1.master"**。

>  注释：母版页也能够包含代码，允许动态的内容。

内容页实例

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
<h2>Individual Content</h2>
<p>Paragraph 1</p>
<p>Paragraph 2</p>
</asp:Content>
```

上面的内容页是站点中独立的内容页中的一个。

@ Page 指令定义它为一个标准的内容页。

内容页包含内容标签 **<asp:Content>**，该标签引用了母版页（ContentPlaceHolderId="CPH1"）。

这个内容页被保存为 **"mypage1.aspx"**。

当用户请求该页面时，ASP.NET 就会将母版页与内容页进行合并。

点击这里显示 mypage1.aspx

>  注释：内容文本必须位于 <asp:Content> 标签内部。标签外的内容文本是不允许的。

带控件的内容页

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
<h2>W3CSchool</h2>
<form runat="server">
<asp:TextBox id="textbox1" runat="server" />
<asp:Button id="button1" runat="server" text="Button" />
</form>
</asp:Content>
```

上面的内容页演示了如何把 .NET 控件插入内容页，就像插入一个普通的页面中。

点击这里显示 mypage2.aspx

ASP.NET Web Forms - 导航

ASP.NET 带有内建的导航控件。

网站导航

维护大型网站的菜单是困难而且费时的。

在 ASP.NET 中，菜单可存储在文件中，这样易于维护。文件通常名为 **web.sitemap**，并且被存放在网站的根目录下。

此外，ASP.NET 有三个心的导航控件：

- Dynamic menus
- TreeViews
- Site Map Path

Sitemap 文件

在本教程中，使用下面的 sitemap 文件：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<siteMap>
<siteMapNode title="Home" url="/aspnet/w3home.aspx">
<siteMapNode title="Services" url="/aspnet/w3services.aspx">
<siteMapNode title="Training" url="/aspnet/w3training.aspx"/>
<siteMapNode title="Support" url="/aspnet/w3support.aspx"/>
</siteMapNode>
</siteMapNode>
</siteMap>
```

创建 sitemap 文件的规则：

- XML 文件必须包含 围绕内容的 <siteMap> 标签
- <siteMap> 标签只能有一个 <siteMapNode> 子节点（ "home" 页面）
- 每个 <siteMapNode> 可以有多个子节点（网页）
- 每个 <siteMapNode> 带有定义页面标题和 URL 的属性

>  注释：sitemap 文件必须位于站点根目录下，URL 属性必须相对于该根目录。

动态菜单

<asp:Menu> 控件可显示标准的站点导航菜单。

代码实例：

```
<asp:SiteMapDataSource id="nav1" runat="server" />

<form runat="server">
<asp:Menu runat="server" DataSourceId="nav1" />
</form>
```

上面实例中的 **<asp:Menu>** 控件是一个供服务器创建导航菜单的占位符。

控件的数据源由 **DataSourceId** 属性定义。 **id="nav1"** 把数据源连接到 **<asp:SiteMapDataSource>** 控件。

<asp:SiteMapDataSource> 控件自动连接默认的 sitemap 文件（**web.sitemap**）。

TreeView

<asp:TreeView> 控件可显示多级导航菜单。

这种菜单看上去像一棵带有枝叶的树，可通过 + 或 - 符号来打开或关闭。

代码实例：

```
<asp:SiteMapDataSource id="nav1" runat="server" />

<form runat="server">
<asp:TreeView runat="server" DataSourceId="nav1" />
</form>
```

上面实例中的 **<asp:TreeView>** 控件是一个供服务器创建导航菜单的占位符。

控件的数据源由 **DataSourceId** 属性定义。 **id="nav1"** 把数据源连接到 **<asp:SiteMapDataSource>** 控件。

<asp:SiteMapDataSource> 控件自动连接默认的 sitemap 文件（**web.sitemap**）。

SiteMapPath

SiteMapPath 控件可显示指向当前页面的指针（导航路径）。该路径显示为指向上级页面的可点击链接。

与 TreeView 和 Menu 控件不同，SiteMapPath 控件不使用 SiteMapDataSource。SiteMapPath 控件默认使用 web.sitemap 文件。

> 提示：如果 SiteMapPath 没有正确显示，很可能是由于 web.sitemap 文件中存在 URL 错误（打印错误）。

代码实例：

```
<form runat="server">
<asp:SiteMapPath runat="server" />
</form>
```

上面实例中的 **<asp:SiteMapPath>** 控件是一个供服务器创建导航菜单的占位符。

Web Pages 参考手册

ASP.NET Web Pages - 类

ASP.NET 类参考手册

方法	描述
AsBool(), AsBool(true|false)	转换字符串值为布尔值（true/false）。如果字 不能转换为true/false， 回 false 或者其他规定的
AsDateTime(), AsDateTime(*value*)	转换字符串值为日期/时 返回 DateTime。如果字 不能转换为日期/时间， 回 MinValue 或者其他规 值。
AsDecimal(), AsDecimal(*value*)	转换字符串值为十进制 如果字符串不能转换为 制值，则返回 0.0 或者 规定的值。
AsFloat(), AsFloat(*value*)	转换字符串值为浮点数 果字符串不能转换为浮 数，则返回 0.0 或者其 定的值。
AsInt(), AsInt(*value*)	转换字符串值为整数。 字符串不能转换成整数， 返回 0 或者其他规定的
Href(*path [, param1 [, param2]]*)	从带有可选的附加路径 的本地文件路径创建一 览器兼容的 URL。
Html.Raw(*value*)	Renders *value* 呈现为 标记，而不是呈现为 H 编码输出。
IsBool(), IsDateTime(), IsDecimal(), IsFloat(), IsInt()	如果该值可以从字符串 为指定的类型，则返回 true。
IsEmpty()	如果对象或者变量没有 则返回 true。
IsPost	如果请求是 POST，则 true。（初始请求通常是 GET。）
Layout	规定布局页面的路径应 此页面。

PageData[*key*], PageData[*index*], Page	在当前请求的页面、布 面、部分页面之间包含 数据。您可以使用动态 来对相同的数据进行属 问。
RenderBody()	(Layout pages) 呈现没 布局页面任何命名区域 容页的内容Renders the content of a content pa that is not in any name sections.
RenderPage(*path*, *values*) RenderPage(*path[, param1 [, param2]]*)	呈现使用了规定的路径 选的额外数据的内容页 可以通过 position（实 或者 key（实例 2）从 PageData 获取额外参 值。
RenderSection(*sectionName [, required* = true|false*]*)	(Layout pages) 呈现一 名字的内容区域。设置 *required* 让一个区域为 非可选的。
Request.Cookies[*key*]	获取或者设置 HTTP co 的值。
Request.Files[*key*]	Gets 在当前请求中上传 件。
Request.Form[*key*]	获取在表单中 post 的数 （作为字符串）。 Request.Form 和 Request.QueryString 者 [*key*] 检查。
Request.QueryString[*key*]	获取 URL 查询字符串中 的数据。Request.Form Request.QueryString 者 [*key*] 检查。
Request.Unvalidated(*key*) Request.Unvalidated().QueryString|Form|Cookies|Headers[*key*]	有选择地禁用请求验证 单元素、查询字符串值、 cookie、header 值）。 验证默认是开启的，防 户提交标记或者其他潜 危险内容。
Response.AddHeader(*name*, *value*)	在应答中添加一个 HTT 务器响应头。
Response.OutputCache(*seconds [, sliding] [, varyByParams]*)	Caches 在指定时间的 出缓存。设置 *sliding* 来 每个页面的访问超时时 设置 *varyByParams* 为

	页面的每个不同的查询 串缓存不同版本的页面。
Response.Redirect(*path*)	重定向浏览器请求到一 的位置。
Response.SetStatus(*httpStatusCode*)	设置HTTP状态代码发送 览器。
Response.WriteBinary(*data [, mimetype]*)	写入 *data* 内容响应可选 MIME类型。
Response.WriteFile(*file*)	写入文件内容响应。
@section(*sectionName*) { *content* }	（布局页面）定义一个 字的内容区域。
Server.HtmlDecode(*htmlText*)	解码一个HTML编码的 串。
Server.HtmlEncode(*text*)	为呈现在 HTML 标记中 符串编码。
Server.MapPath(*virtualPath*)	为指定的虚拟路径返回 器的物理路径。
Server.UrlDecode(*urlText*)	解码URL文本。
Server.UrlEncode(*text*)	URL文本编码。
Session[*key*]	获取或设置一个存在的 直到用户关闭浏览器。
ToString()	显示一个用字符串表示 象的值。
UrlData[*index*]	从 URL 获取额外的数据 如，*/MyPage/ExtraDat*

ASP.NET Web Pages - WebSecurity 对象

描述

WebSecurity 对象提供 ASP.NET Web Pages 应用程序的安全性和认证。

通过 WebSecurity 对象，您可以创建用户帐户，登录和注销用户，重置或者更改密码，以及其他更多与安全性相关的功能。

WebSecurity 对象参考手册 - 属性

属性	描述
CurrentUserId	获取当前登录用户的 ID。
CurrentUserName	获取当前登录用户的名称。
HasUserId	如果当前有用户 ID，则返回 true。
IsAuthenticated	如果当前用户是登录的，则返回 true。

WebSecurity 对象参考手册 - 方法

方法	描述
ChangePassword()	为指定的用户更改密码。
ConfirmAccount()	使用帐户确认令牌确认帐户。
CreateAccount()	创建一个新的用户帐户。
CreateUserAndAccount()	创建一个新的用户帐户。
GeneratePasswordResetToken()	生成一个密码重置令牌，可以在电子邮件中发送给用户以便用户可以重设密码。
GetCreateDate()	获取指定会员创建的时间。
GetPasswordChangeDate()	获取密码变更的日期和时间。
GetUserId()	根据用户名称获取用户 ID。
InitializeDatabaseConnection()	初始化 WebSecurity 系统（数据库）。
IsConfirmed()	检查用户是否已被确认。如果已确认，则返回 true。（例如，可通过电子邮件进行确认。）
IsCurrentUser()	检查当前用户的名称是否与指定用户名匹配。如果匹配，则返回 true。
Login()	设置身份验证令牌，登录用户。
Logout()	移除身份验证令牌，注销用户。
RequireAuthenticatedUser()	如果用户未通过身份验证，则设置 HTTP 状态为 401（未经授权）。
RequireRoles()	如果当前用户不是指定角色的成员，则设置 HTTP 状态为 401（未经授权）。
RequireUser()	如果当前用户不是指定用户名的用户，则设置 HTTP 状态为 401（未经授权）。
ResetPassword()	如果密码重置令牌是有效的，改变用户的密码为新密码。
UserExists()	检查指定的用户是否存在。

技术数据

名称	值
Class	WebMatrix.WebData.WebSecurity
Namespace	WebMatrix.WebData
Assembly	WebMatrix.WebData.dll

初始化 WebSecurity 数据库

如果您想在您的代码中使用 WebSecurity 对象，首先您必须创建或者初始化 WebSecurity 数据库。

在您的 Web 根目录下，创建一个名为 **_AppStart.cshtml** 的页面（如果已存在，则直接编辑页面）。

将下面的代码复制到文件中：

_AppStart.cshtml

```
@{
WebSecurity.InitializeDatabaseConnection("Users", "UserProfile", "UserId", "Email", true)
}
```

上面的代码将在每次网站（应用程序）启动时运行。它初始化了 WebSecurity 数据库。

"Users" 是 WebSecurity 数据库（Users.sdf）的名称。

"UserProfile" 是包含用户配置信息的数据库表的名称。

"UserId" 是包含用户 ID（主键）的列的名称。

"Email" 是包含用户名的列的名称。

最后一个参数 **true** 是一个布尔值，表示如果用户配置表和会员表不存在，则会自动创建表。如果不想自动创建表，应设置参数为 **false**。

> 虽然 **true** 表示自动创建数据库 表，但是数据库不会被自动创建。所以数据库必须存在。

WebSecurity 数据库

UserProfile 表为每个用户创建保存一条记录，用户 ID（主键）和用户名字（email）：

UserId	Email
1	john@johnson.net
2	peter@peterson.com
3	lars@larson.eut

Membership 表包含会员信息，比如用户是什么时候创建的，该会员是否已认证，会员是什么时候认证的，等等。

具体如下所示（一些列不显示）：

User Id	Create Date	Confirmation Token	Is Confirmed	Last Password Failure	Password	F
1	12.04.2012 16:12:17	NULL	True	NULL	AFNQhWfy....	1 1

注释：如果您想看到所有的列和内容，请打开数据库，看看里边的每个表。

简单的会员配置

在您使用 WebSecurity 对象时，如果您的站点没有配置使用 ASP.NET Web Pages 会员系统 **SimpleMembership**，可能会报错。

如果托管服务提供商的服务器的配置与您本地服务器的配置不同，也可能会报错。为了解决这个问题，请在网站的 Web.config 文件中添加以下元素：

```
<appSettings>
<add key="enableSimpleMembership" value="true" />
</appSettings>
```

ASP.NET Web Pages - Database 对象

ASP.NET Database 对象参考手册

方法	描述
Database.Execute(*SQLstatement [, parameters])*	执行 SQL 语句 *SQLstatement*（带可选参数），比如 INSERT、DELETE 或者 UPDATE， 并且返回受影响的记录统计。
Database.GetLastInsertId()	返回最近插入行的标识列。
Database.Open(*filename*) Database.Open(*connectionStringName*)	使用 *Web.config* 文件中的连接字符串打开指定的数据库文件或者指定的数据库。
Database.OpenConnectionString(*connectionString*)	使用连接字符串打开一个数据库。（与 Database.Open 的差异是，Database.Open 使用的是连接字符串的名称，连接字符串的值在其他地方配置。）
Database.Query(*SQLstatement[, parameters]*)	使用 SQL 语句 *SQLstatement*（带可选参数）查询数据库，并返回结果集合。
Database.QuerySingle(*SQLstatement [, parameters]*)	执行 SQL 语句 *SQLstatement*（带可选参数），并返回单条记录。
Database.QueryValue(*SQLstatement [, parameters]*)	执行 SQL 语句 *SQLstatement*（带可选参数），并返回单个值。

ASP.NET Web Pages - WebMail 对象

通过 WebMail 对象，您可以很容易地从网页上发送电子邮件。

描述

WebMail 对象为 ASP.NET Web Pages 提供了使用 SMTP（Simple Mail Transfer Protocol 简单邮件传输协议）发送邮件的功能。

实例

请查看 WebPages Email 章节中的实例。

WebMail 对象参考手册 - 属性

属性	描述
SmtpServer	用于发送电子邮件的 SMTP 服务器的名称。
SmtpPort	服务器用来发送 SMTP 电子邮件的端口。
EnableSsl	如果服务器使用 SSL（Secure Socket Layer 安全套接层）加密，则值为 true。
UserName	用于发送电子邮件的 SMTP 电子邮件账户的名称。
Password	SMTP 电子邮件账户的密码。
From	在发件地址栏显示的电子邮件（通常与 UserName 相同）。

WebMail 对象参考手册 - 方法

方法	描述
Send()	向 SMTP 服务器发送需要传送的电子邮件信息。

Send() 方法有以下参数：

参数	类型	描述
to	String	收件人（用分号分隔）
subject	String	邮件主题
body	String	邮件正文

Send() 方法有以下可选参数：

参数	类型	描述
from	String	发件人
cc	String	需要抄送的电子邮件地址（用分号分隔）
filesToAttach	Collection	附件名
isBodyHtml	Boolean	如果邮件正文是 HTML 格式的，则为 true
additionalHeaders	Collection	附加的标题

技术数据

名称	值
Class	System.Web.Helpers.WebMail
Namespace	System.Web.Helpers
Assembly	System.Web.Helpers.dll

初始化 **WebMail** 帮助器

要使用 WebMail 帮助器，您必须能访问 SMTP 服务器。SMTP 是电子邮件的"输出"部分。如果您使用的是虚拟主机，您可能已经知道 SMTP 服务器的名称。如果您使用的是公司网络工作，您公司的 IT 部门会给您一个名称。如果您是在家工作，你也许可以使用普通的电子邮件服务提供商。

为了发送一封电子邮件，您将需要：

- SMTP 服务器的名称
- 端口号（通常是 25 ）
- 电子邮件的用户名
- 电子邮件的密码

在您的 Web 根目录下，创建一个名为 **_AppStart.cshtml** 的页面（如果已存在，则直接编辑页面）。

将下面的代码复制到文件中：

_AppStart.cshtml

```
@{
WebMail.SmtpServer = "smtp.example.com";
WebMail.SmtpPort = 25;
WebMail.EnableSsl = false;
WebMail.UserName = "support@example.com";
WebMail.Password = "password";
WebMail.From = "john@example.com"
}
```

上面的代码将在每次网站（应用程序）启动时运行。它对 **WebMail** 对象赋了初始值。

请替换：

将 **smtp.example.com** 替换成您要用来发送电子邮件的 SMTP 服务器的名称。

将 **25** 替换成服务器用来发送 SMTP 事务（电子邮件）的端口号。

如果服务器使用 SSL（Secure Socket Layer 安全套接层）加密，请将 **false** 替换成 true。

将 **support@example.com** 替换成用来发送电子邮件的 SMTP 电子邮件账户的名称。

将 **password** 替换成 SMTP 电子邮件账户的密码。

将 **john@example** 替换成显示在发件地址栏中的电子邮件。

> 在您的 AppStart 文件中，您不需要启动 **WebMail** 对象，但是在调用 **WebMail.Send()** 方法之前，您必须设置这些属性。

ASP.NET Web Pages - 更多帮助器

ASP.NET 帮助器 - 对象参考手册

Analytics 对象参考手册（Google）

Helper	描述
Analytics.GetGoogleHtml(*webPropertyId*)	为指定的 ID 呈现 Google Analytics JavaScript 代码。
Analytics.GetStatCounterHtml(*project*, *security*)	为指定的项目呈现 StatCounter Analytics JavaScript 代码。
Analytics.GetYahooHtml(*account*)	为指定的账号呈现 Yahoo Analytics JavaScript 代码。

Bing 对象参考手册

Helper	描述
Bing.SearchBox(*[boxWidth]*)	给 Bing 传递搜索。您可以设置 Bing.SiteUrl 和 Bing.SiteTitle 属性来设定站点搜索和搜索框的标题，通常是在 *_AppStart* 页面设置这些属性。
Bing.AdvancedSearchBox(*[, boxWidth] [, resultWidth] [, resultHeight] [, themeColor] [, locale]*)	用可选的格式显示 Bing 搜索结果在页面上。您可以设置 Bing.SiteUrl 和 Bing.SiteTitle 属性来设定站点搜索和搜索框的标题，通常是在 *_AppStart* 页面设置这些属性。

Chart 对象参考手册

Helper	描述
Chart(*width*, *height [, template] [, templatePath]*)	初始化图表。
Chart.AddLegend(*[title] [, name]*)	给图表添加一个图例。
Chart.AddSeries(*[name] [, chartType] [, chartArea] [, axisLabel] [, legend] [, markerStep] [, xValue] [, xField] [, yValues] [, yFields] [, options]*)	给图表添加一系列数据。

Crypto 对象参考手册

Helper	描述
Crypto.Hash(*string [, algorithm]*) Crypto.Hash(*bytes [, algorithm]*)	返回指定数据的哈希。默认算法是 sha256。

Facebook 对象参考手册

Helper	描述
Facebook.LikeButton(*href [, buttonLayout] [, showFaces] [, width] [, height] [, action] [, font] [, colorScheme] [, refLabel]*)	让 Facebook 用户连接到网页。

FileUpload 对象参考手册

Helper	描述
FileUpload.GetHtml(*[initialNumberOfFiles] [, allowMoreFilesToBeAdded] [, includeFormTag] [, addText] [, uploadText]*)	为上传文件呈现 UI。

GamerCard 对象参考手册

Helper	描述
GamerCard.GetHtml(*gamerTag*)	呈现指定的 Xbox gamer 标签。

Gravatar 对象参考手册

Helper	描述
Gravatar.GetHtml(*email [, imageSize] [, defaultImage] [, rating] [, imageExtension] [, attributes]*)	为指定的电子邮件地址呈现 Gravatar 图像。

Json 对象参考手册

Helper	描述
Json.Encode(*object*)	用 JavaScript Object Notation (JSON) 把数据对象转换为字符串。
Json.Decode(*string*)	转换 JSON 编码的输入字符串为您指定的数据对象。

LinkShare 对象参考手册

Helper	描述
LinkShare.GetHtml(*pageTitle [, pageLinkBack] [, twitterUserName] [, additionalTweetText] [, linkSites]*)	使用指定的标题和可选的 URL 呈现社会网络链接。

ModelState 对象参考手册

Helper	描述
ModelStateDictionary.AddError(*key*, *errorMessage*)	关联错误信息和一个表单域。使用 ModelState 帮助器访问成员。
ModelStateDictionary.AddFormError(*errorMessage*)	关联错误信息和一个表单。使用 ModelState 帮助器访问成员。
ModelStateDictionary.IsValid	如果没有验证错误，返回 true。使用 ModelState 帮助器访问成员。

ObjectInfo 对象参考手册

Helper	描述
ObjectInfo.Print(*value [, depth] [, enumerationLength]*)	呈现一个对象和所有子对象的属性和值。

Recaptcha 对象参考手册

Helper	描述
Recaptcha.GetHtml(*[, publicKey] [, theme] [, language] [, tabIndex]*)	呈现 reCAPTCHA 验证测试。
ReCaptcha.PublicKey ReCaptcha.PrivateKey	设置 reCAPTCHA 服务的公共和私有密钥。通常是在 *_AppStart* 页面设置这些属性。
ReCaptcha.Validate(*[, privateKey]*)	返回 reCAPTCHA 测试结果。
ServerInfo.GetHtml()	Renders 呈现有关 ASP.NET Web Pages 的状态信息。

Twitter 对象参考手册

Helper	描述
Twitter.Profile(*twitterUserName*)	为指定的用户呈现 Twitter 流。
Twitter.Search(*searchQuery*)	为指定的搜索文本呈现 Twitter 流。

Video 对象参考手册

Helper	描述
Video.Flash(*filename [, width, height]*)	为指定的文件呈现宽度和高度可选的 Flash 视频播放。
Video.MediaPlayer(*filename [, width, height]*)	为指定的文件呈现宽度和高度可选 的 Windows Media 播放器。
Video.Silverlight(*filename*, *width*, *height*)	为指定的 *.xap* 文件呈现所需的宽度和高度 的 Silverlight 播放器。

WebCache 对象参考手册

Helper	描述
WebCache.Get(*key*)	通过 *key* 返回指定的对象，如果对象未找到则返回 null。
WebCache.Remove(*key*)	通过 *key* 从缓存中删除指定的对象。
WebCache.Set(*key*, *value [, minutesToCache] [, slidingExpiration]*)	通过 *key* 把 *value* 放置到指定名称的缓存中。

WebGrid 对象参考手册

Helper	描述
WebGrid(*data*)	Creates a 使用查询数据创建一个新的 WebGrid 对象。
WebGrid.GetHtml()	Renders markup 显示数据在 HTML 表格中。
WebGrid.Pager()	为 WebGrid 对象呈现一个页面。

WebImage 对象参考手册

Helper	描述
WebImage(*path*)	从指定的路径加载一个图像。
WebImage.AddImagesWatermark(*image*)	为指定图像加水印。
WebImage.AddTextWatermark(*text*)	为图像添加指定文本。
WebImage.FlipHorizontal() WebImage.FlipVertical()	水平/垂直翻转图像
WebImage.GetImageFromRequest()	当图像被传送到一个文件上传页面时，加载图像。
WebImage.Resize(*width*, *height*)	调整图像大小。
WebImage.RotateLeft() WebImage.RotateRight()	向左或向右旋转图像。
WebImage.Save(*path [, imageFormat]*)	保存图像到指定路径。

ASP.NET MVC - 参考手册

类

类	描述
AcceptVerbsAttribute	表示一个特性，该特性指定操作方法将响应的 HTTP 谓词。
ActionDescriptor	提供有关操作方法的信息，比如操作方法的名称、控制器、参数、特性和筛选器。
ActionExecutedContext	提供 ActionFilterAttribute 类的 ActionExecuted 方法的上下文。
ActionExecutingContext	提供 ActionFilterAttribute 类的 ActionExecuting 方法的上下文。
ActionFilterAttribute	表示筛选器特性的基类。
ActionMethodSelectorAttribute	表示一个用于影响操作方法选择的特性。
ActionNameAttribute	表示一个用于操作的名称的特性。
ActionNameSelectorAttribute	表示一个可影响操作方法选择的特性。
ActionResult	封装一个操作方法的结果并用于代表该操作方法执行框架级操作。
AdditionalMetadataAttribute	提供一个类，该类实现 IMetadataAware 接口以支持其他元数据。
AjaxHelper	表示支持在视图中呈现 AJAX 方案中的 HTML。
AjaxHelper(TModel)	表示支持在强类型视图中呈现 AJAX 方案中的 HTML。
AjaxRequestExtensions	表示一个类，该类对 HttpRequestBase 类进行了扩展，在其中添加了确定 HTTP 请求是否为 AJAX 请求的功能。
AllowHtmlAttribute	通过跳过属性的请求验证，允许请求在模型绑定过程中包含 HTML 标记。（强烈建议应用程序显式检查所有禁用请求验证的模型，以防止脚本攻击。）
AreaRegistration	提供在一个 ASP.NET MVC 应用程序内注册一个或多个区域的方式。
AreaRegistrationContext	对在 ASP.NET MVC 应用程序内注册某个区域时所需的信息进行封装。
AssociatedMetadataProvider	提供用于实现元数据提供程序的抽象类。

AssociatedValidatorProvider	为用于实现验证提供程序的类提供抽象类。
AsyncController	为异步控制器提供基类。
AsyncTimeoutAttribute	表示一个特性，该特性用于设置异步方法的超时值（以毫秒为单位）。
AuthorizationContext	对使用 AuthorizeAttribute 特性时所需的信息进行封装。
AuthorizeAttribute	表示一个特性，该特性用于限制调用方对操作方法的访问。
BindAttribute	表示一个特性，该特性用于提供有关应如何进行模型绑定到参数的详细信息。
BuildManagerCompiledView	表示在视图引擎呈现视图之前由 BuildManager 类编译的视图的基类。
BuildManagerViewEngine	为视图引擎提供基类。
ByteArrayModelBinder	映射浏览器请求到字节数组。
ChildActionOnlyAttribute	表示一个特性，该特性用于指示操作方法只应作为子操作进行调用。
ChildActionValueProvider	表示子操作中的值的值提供程序。
ChildActionValueProviderFactory	表示用于为子操作创建值提供程序对象的工厂。
ClientDataTypeModelValidatorProvider	返回客户端数据类型模型验证程序。
CompareAttribute	提供用于比较某个模型的两个属性的特性。
ContentResult	表示用户定义的内容类型，该类型是操作方法的结果。
Controller	提供用于响应对 ASP.NET MVC 网站所进行的 HTTP 请求的方法。
ControllerActionInvoker	表示一个类，该类负责调用控制器的操作方法。
ControllerBase	表示所有 MVC 控制器的基类。
ControllerBuilder	表示一个类，该类负责动态生成控制器。
ControllerContext	封装有关与指定的 RouteBase 和 ControllerBase 实例匹配的 HTTP 请求的信息。
ControllerDescriptor	封装描述控制器的信息，比如控制器的名称、类型和操作。
ControllerInstanceFilterProvider	将控制器添加到 FilterProviderCollection 实例。

CustomModelBinderAttribute	表示一个调用自定义模型联编程序的特性。
DataAnnotationsModelMetadata	为数据模型的公共元数据、DataAnnotationsModelMetadataProvider 类和 DataAnnotationsModelValidator 类提供容器。
DataAnnotationsModelMetadataProvider	实现 ASP.NET MVC 的默认模型元数据提供程序。
DataAnnotationsModelValidator	提供模型验证程序。
DataAnnotationsModelValidator(TAttribute)	为指定的验证类型提供模型验证程序。
DataAnnotationsModelValidatorProvider	实现 ASP.NET MVC 的默认验证提供程序。
DataErrorInfoModelValidatorProvider	为错误信息模型验证程序提供容器。
DefaultControllerFactory	表示默认情况下已注册的控制器工厂。
DefaultModelBinder	映射浏览器请求到数据对象。该类提供模型联编程序的具体实现。
DefaultViewLocationCache	表示视图位置的内存缓存。
DependencyResolver	为实现 IDependencyResolver 或公共服务定位器 IServiceLocator 接口的依赖关系解析程序提供一个注册点。
DependencyResolverExtensions	提供 GetService 和 GetServices 的类型安全实现。
DictionaryValueProvider(TValue)	表示值提供程序的基类，这些值提供程序的值来自实现 IDictionary(TKey, TValue) 接口的集合。
EmptyModelMetadataProvider	为不需要元数据的数据模型提供空的元数据提供程序。
EmptyModelValidatorProvider	为不需要验证程序的模型提供空的验证提供程序。
EmptyResult	表示一个不执行任何操作的结果，比如一个不返回任何内容的控制器操作方法。
ExceptionContext	P提供使用 HandleErrorAttribute 类的上下文。
ExpressionHelper	提供用于从表达式中获取模型名称的帮助器类。
FieldValidationMetadata	为客户端字段验证元数据提供容器。
FileContentResult	将二进制文件的内容发送到响应。
FilePathResult	将文件的内容发送到响应。

FileResult	表示一个用于将二进制文件内容发送到响应的基类。
FileStreamResult	使用 Stream 实例将二进制内容发送到响应。
Filter	表示一个元数据类，它包含对一个或多个筛选器接口的实现、筛选器顺序和筛选器范围的引用。
FilterAttribute	表示操作和结果筛选器特性的基类。
FilterAttributeFilterProvider	定义筛选器特性的筛选器提供程序。
FilterInfo	封装有关可用的操作筛选器的信息。
FilterProviderCollection	表示应用程序的筛选器提供程序的集合。
FilterProviders	为筛选器提供一个注册点。
FormCollection	包含应用程序的表单值提供程序。
FormContext	对验证和处理 HTML 表单中的输入数据所需的信息进行封装。
FormValueProvider	表示 NameValueCollection 对象中包含的表单值的值提供程序。
FormValueProviderFactory	表示一个类，该类负责创建表单值提供程序对象的新实例。
GlobalFilterCollection	表示一个包含所有全局筛选器的类。
GlobalFilters	表示全局筛选器集合。
HandleErrorAttribute	表示一个特性，该特性用于处理由操作方法引发的异常。
HandleErrorInfo	封装有关处理由操作方法引发的错误的信息。
HiddenInputAttribute	表示一个特性，该特性用于指示是否应将属性值或字段值呈现为隐藏的 input 元素。
HtmlHelper	表示支持在视图中呈现 HTML 控件。
HtmlHelper(TModel)	表示支持在强类型视图中呈现 HTML 控件。
HttpDeleteAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP DELETE 请求。
HttpFileCollectionValueProvider	表示要用于来自 HTTP 文件集合的值的值提供程序。
HttpFileCollectionValueProviderFactory	表示一个类，该类负责创建 HTTP 文件集合值提供程序对象的新实例。

HttpGetAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP GET 请求。
HttpNotFoundResult	定义一个用于指示未找到所请求资源的对象。
HttpPostAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP POST 请求。
HttpPostedFileBaseModelBinder	将模型绑定到已发布的文件。
HttpPutAttribute	表示一个特性，该特性用于限制操作方法，以便该方法仅处理 HTTP PUT 请求。
HttpRequestExtensions	扩展 HttpRequestBase 类，该类包含客户端在 Web 请求中发送的 HTTP 值。
HttpStatusCodeResult	提供一种用于返回带特定 HTTP 响应状态代码和说明的操作结果的方法。
HttpUnauthorizedResult	表示未经授权的 HTTP 请求的结果。
JavaScriptResult	将 JavaScript 内容发送到响应。
JsonResult	表示一个类，该类用于将 JSON 格式的内容发送到响应。
JsonValueProviderFactory	启用操作方法以发送和接收 JSON 格式的文本，并将 JSON 文本以模型绑定方式传递给操作方法的参数。
LinqBinaryModelBinder	映射浏览器请求到 LINQ Binary 对象。
ModelBinderAttribute	表示一个特性，该特性用于将模型类型关联到模型-生成器类型。
ModelBinderDictionary	表示一个类，该类包含应用程序的所有模型联编程序（按联编程序类型列出）。
ModelBinderProviderCollection	为模型联编程序提供程序提供一个容器。
ModelBinderProviders	为模型联编程序提供程序提供一个容器。
ModelBinders	提供对应用程序的模型联编程序的全局访问。
ModelBindingContext	提供运行模型联编程序的上下文。
ModelClientValidationEqualToRule	为发送到浏览器的相等验证规则提供一个容器。
ModelClientValidationRangeRule	为发送到浏览器的范围验证规则提供一个容器。
ModelClientValidationRegexRule	为发送到浏览器的正则表达式客户端验证规则提供一个容器。
ModelClientValidationRemoteRule	为发送到浏览器的远程验证规则提供一个容器。

ModelClientValidationRequiredRule	为必填字段的客户端验证提供一个容器。
ModelClientValidationRule	为发送到浏览器的客户端验证规则提供一个基类容器。
ModelClientValidationStringLengthRule	为发送到浏览器的字符串长度验证规则提供一个容器。
ModelError	表示在模型绑定期间发生的错误。
ModelErrorCollection	ModelError 实例的集合。
ModelMetadata	为数据模型的公共元数据、ModelMetadataProvider 类和 ModelValidator 类提供容器。
ModelMetadataProvider	为自定义元数据提供程序提供抽象基类。
ModelMetadataProviders	为当前的 ModelMetadataProvider 实例提供容器。
ModelState	将模型绑定的状态封装到操作方法参数的一个属性或操作方法参数本身。
ModelStateDictionary	表示将已发送表单绑定到操作方法（其中包括验证信息）的尝试的状态。
ModelValidationResult	为验证结果提供容器。
ModelValidator	提供用于实现验证逻辑的基类。
ModelValidatorProvider	为模型提供验证程序的列表。
ModelValidatorProviderCollection	为验证提供程序的列表提供一个容器。
ModelValidatorProviders	为当前验证提供程序提供容器。
MultiSelectList	表示一个项列表，用户可从该列表中选择多个项。
MvcFilter	在派生类中实现时，提供一个元数据类，它包含对一个或多个筛选器接口的实现、筛选器顺序和筛选器范围的引用。
MvcHandler	选择将处理 HTTP 请求的控制器。
MvcHtmlString	表示不应再次进行编码的 HTML 编码的字符串。
MvcHttpHandler	验证并处理 HTTP 请求。
MvcRouteHandler	创建一个实现 IHttpHandler 接口的对象并向该对象传递请求上下文。
MvcWebRazorHostFactory	创建 MvcWebPageRazorHost 文件的实例。
NameValueCollectionExtensions	扩展 NameValueCollection 对象，以便能够将集合复制到指定字典。

NameValueCollectionValueProvider	表示值提供程序的基类，这些值提供程序的值来自 NameValueCollection 对象。
NoAsyncTimeoutAttribute	为 AsyncTimeoutAttribute 特性提供便利包装。
NonActionAttribute	表示一个特性，该特性用于指示控制器方法不是操作方法。
OutputCacheAttribute	表示一个特性，该特性用于标记将缓存其输出的操作方法。
ParameterBindingInfo	封装与将操作方法参数绑定到数据模型相关的信息。
ParameterDescriptor	包含描述参数的信息。
PartialViewResult	表示一个用于将部分视图发送到响应的基类。
PreApplicationStartCode	为 ASP.NET Razor 应用程序预启动代码提供注册点。
QueryStringValueProvider	表示 NameValueCollection 对象中包含的查询字符串的值提供程序。
QueryStringValueProviderFactory	表示一个类，该类负责创建查询字符串值提供程序对象的新实例。
RangeAttributeAdapter	提供 RangeAttribute 特性的适配器。
RazorView	表示用于创建具有 Razor 语法的视图的类。
RazorViewEngine	表示一个用于呈现使用 ASP.NET Razor 语法的 Web 页面的视图引擎。
RedirectResult	通过重定向到指定的 URI 来控制对应用程序操作的处理。
RedirectToRouteResult	表示使用指定的路由值字典来执行重定向的结果。
ReflectedActionDescriptor	包含描述反射的操作方法的信息。
ReflectedControllerDescriptor	包含描述反射的控制器的信息。
ReflectedParameterDescriptor	包含描述反射的操作方法参数的信息。
RegularExpressionAttributeAdapter	提供 RegularExpressionAttribute 特性的适配器。
RemoteAttribute	提供使用 jQuery 验证插件远程验证程序的特性。
RequiredAttributeAdapter	提供 RequiredAttributeAttribute 特性的适配器。
RequireHttpsAttribute	表示一个特性，该特性用于强制通过 HTTPS 重新发送不安全的 HTTP 请求。

ResultExecutedContext	提供 ActionFilterAttribute 类的 OnResultExecuted 方法的上下文。
ResultExecutingContext	提供 ActionFilterAttribute 类的 OnResultExecuting 方法的上下文。
RouteCollectionExtensions	扩展 RouteCollection 对象以进行 MVC 路由。
RouteDataValueProvider	表示实现 IDictionary(TKey, TValue) 接口的对象中包含的路由数据的值提供程序。
RouteDataValueProviderFactory	表示用来创建路由数据值提供程序对象的工厂。
SelectList	表示一个列表，用户可从该列表中选择一个项。
SelectListItem	表示 SelectList 类的实例中的选定项。
SessionStateAttribute	指定控制器的会话状态。
SessionStateTempDataProvider	为当前 TempDataDictionary 对象提供会话状态数据。
StringLengthAttributeAdapter	提供 StringLengthAttribute 特性的适配器。
TempDataDictionary	表示仅从一个请求保持到下一个请求的数据集。
TemplateInfo	封装有关当前模板上下文的信息。
UrlHelper	包含用于为应用程序内的 ASP.NET MVC 生成 URL 的方法。
UrlParameter	表示路由过程中 MvcHandler 类使用的可选参数。
ValidatableObjectAdapter	提供可验证的对象适配器。
ValidateAntiForgeryTokenAttribute	表示用于阻止伪造请求的特性。
ValidateInputAttribute	表示一个特性，该特性用于标记必须验证其输入的操作方法。
ValueProviderCollection	表示应用程序的值提供程序对象的集合。
ValueProviderDictionary	已过时。表示应用程序的值提供程序的字典。
ValueProviderFactories	表示值提供程序工厂对象的容器。
ValueProviderFactory	表示用来创建值提供程序对象的工厂。
ValueProviderFactoryCollection	表示应用程序的值提供程序工厂的集合。
ValueProviderResult	表示将一个值（如表单发送的值或查询字符串中的值）绑定到操作方法参数属性或

	绑定到该参数本身的结果。
ViewContext	封装与呈现视图相关的信息。
ViewDataDictionary	表示一个容器，该容器用于在控制器和视图之间传递数据。
ViewDataDictionary(TModel)	表示一个容器，该容器用于在控制器和视图之间传递强类型数据。
ViewDataInfo	对开发模板所使用的当前模板内容和与模板交互的 HTML 帮助器的相关信息进行封装。
ViewEngineCollection	表示对应用程序可用的视图引擎的集合。
ViewEngineResult	表示定位视图引擎的结果。
ViewEngines	表示对应用程序可用的视图引擎的集合。
ViewMasterPage	表示生成母版视图页所需的信息。
ViewMasterPage(TModel)	表示生成强类型母版视图页所需的信息。
ViewPage	表示将视图呈现为 Web Forms 页所需的属性和方法。
ViewPage(TModel)	表示将强类型视图呈现为 Web Forms 页所需的信息。
ViewResult	表示一个类，该类用于使用由 IViewEngine 对象返回的 IView 实例来呈现视图。
ViewResultBase	表示一个用于为视图提供模型并向响应呈现视图的基类。
ViewStartPage	提供可用于实现视图启动（母版）页的抽象类。
ViewTemplateUserControl	提供 TemplateInfo 对象的容器。
ViewTemplateUserControl(TModel)	提供 TemplateInfo 对象的容器。
ViewType	表示视图的类型。
ViewUserControl	表示生成用户控件所需的信息。
ViewUserControl(TModel)	表示生成强类型用户控件所需的信息。
VirtualPathProviderViewEngine	表示 IViewEngine 接口的抽象基类实现。
WebFormView	表示在 ASP.NET MVC 中生成 Web Forms 页时所需的信息。
WebFormViewEngine	表示一个用于向响应呈现 Web Forms 页的视图引擎。
WebViewPage	表示呈现使用 ASP.NET Razor 语法的视图所需的属性和方法。

WebViewPage(TModel)	表示呈现使用 ASP.NET Razor 语法的视图所需的属性和方法。

接口

接口	描述
IActionFilter	定义操作筛选器中使用的方法。
IActionInvoker	定义操作调用程序的协定，该调用程序用于调用一个操作以响应 HTTP 请求。
IAuthorizationFilter	定义授权筛选器所需的方法。
IClientValidatable	为 ASP.NET MVC 验证框架提供一种用于在运行时发现验证程序是否支持客户端验证的方法。
IController	定义控制器所需的方法。
IControllerActivator	对使用依赖关系注入来实例化控制器的方式进行精细控制。
IControllerFactory	定义控制器工厂所需的方法。
IDependencyResolver	定义可简化服务位置和依赖关系解析的方法。
IExceptionFilter	定义异常筛选器所需的方法。
IFilterProvider	提供用于查找筛选器的接口。
IMetadataAware	提供用于向 AssociatedMetadataProvider 类公开特性的接口。
IModelBinder	定义模型联编程序所需的方法。
IModelBinderProvider	定义用于为实现 IModelBinder 接口的类动态实现模型绑定的方法。
IMvcFilter	定义用于指定筛选器顺序以及是否允许多个筛选器的成员。
IResultFilter	定义结果筛选器所需的方法。
IRouteWithArea	将路由与 ASP.NET MVC 应用程序中的区域关联。
ITempDataProvider	定义临时数据提供程序的协定，这些临时数据提供程序用于存储要在下一个请求中查看的数据。
IUnvalidatedValueProvider	表示一个可跳过请求验证的 IValueProvider 接口。
IValueProvider	定义 ASP.NET MVC 中的值提供程序所需的方法。
IView	定义视图所需的方法。
IViewDataContainer	定义视图数据字典所需的方法。
IViewEngine	定义视图引擎所需的方法。
IViewLocationCache	定义在内存中缓存视图位置所需的方法。
IViewPageActivator	对使用依赖关系注入创建视图页的方式进行精细控制。

Web Forms 参考手册

ASP.NET Web Forms - HTML 服务器控件

HTML 服务器控件是服务器可理解的 HTML 标签。

HTML 服务器控件

ASP.NET 文件中的 HTML 元素，默认是作为文本进行处理的。要想让这些元素可编程，需向 HTML 元素中添加 runat="server" 属性。这个属性表示，该元素将被作为服务器控件进行处理。

注释：所有 HTML 服务器控件必须位于带有 runat="server" 属性的 <form> 标签内！

注释：ASP.NET 要求所有 HTML 元素必须正确关闭和正确嵌套。

HTML 服务器控件	描述
HtmlAnchor	控制 `<a>` HTML 元素
HtmlButton	控制 `<button>` HTML 元素
HtmlForm	控制 `<form>` HTML 元素
HtmlGeneric	控制其他未被具体的 HTML 服务器控件规定的 HTML 元素，比如 `<body>` 、 `<div>` 、 `<span>` 等。
HtmlImage	控制 `<image>` HTML 元素
HtmlInputButton	控制 `<input type="button">` 、 `<input type="submit">` 和 `<input type="reset">` HTML 元素
HtmlInputCheckBox	控制 `<input type="checkbox">` HTML 元素
HtmlInputFile	控制 `<input type="file">` HTML 元素
HtmlInputHidden	控制 `<input type="hidden">` HTML 元素
HtmlInputImage	控制 `<input type="image">` HTML 元素
HtmlInputRadioButton	控制 `<input type="radio">` HTML 元素
HtmlInputText	控制 `<input type="text">` 和 `<input type="password">` HTML 元素
HtmlSelect	控制 `<select>` HTML 元素
HtmlTable	控制 `<table>` HTML 元素
HtmlTableCell	控制 `<td>` 和 `<th>` HTML 元素
HtmlTableRow	控制 `<tr>` HTML 元素
HtmlTextArea	控制 `<textarea>` HTML 元素

ASP.NET Web Forms - Web 服务器控件

Web 服务器控件是服务器可理解的特殊 ASP.NET 标签。

Web 服务器控件

就像 HTML 服务器控件，Web 服务器控件也是在服务器上创建的，它们同样需要 runat="server" 属性才能生效。然而，Web 服务器控件没有必要映射任何已存在的 HTML 元素，它们可以表示更复杂的元素。

创建 Web 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

Web 服务器控件	描述
AdRotator	显示一个图形序列
Button	显示下压按钮
Calendar	显示日历
CalendarDay	calendar 控件中的一天
CheckBox	显示复选框
CheckBoxList	创建多选的复选框组
DataGrid	显示 grid 中数据源的字段
DataList	通过使用模版显示数据源中的项目
DropDownList	创建下拉列表
HyperLink	创建超链接
Image	显示图像
ImageButton	显示可点击的图像
Label	显示可编程的静态内容（使您对其内容应用样式）
LinkButton	创建超链接按钮
ListBox	创建单选或多选的下拉列表
ListItem	创建列表中的一个项目
Literal	显示可编程的静态内容（无法使您对其内容应用样式）
Panel	为其他控件提供容器
PlaceHolder	为由代码添加的控件预留空间
RadioButton	创建单选按钮
RadioButtonList	创建单选按钮组
BulletedList	创建项目符号格式的列表
Repeater	显示绑定到控件的项目的重复列表
Style	设置控件的样式
Table	创建表格
TableCell	创建表格单元格
TableRow	创建表格行
TextBox	创建文本框
Xml	显示 XML 文件或 XSL 转换的结果

ASP.NET Web Forms - Validation 服务器控件

Validation 服务器控件是用来验证用户输入的。

Validation 服务器控件

Validation 服务器控件用于验证输入控件的数据。如果数据未通过验证，则向用户显示错误消息。

创建 Validation 服务器控件的语法是：

```
<asp:control_name id="some_id" runat="server" />
```

Validation 服务器控件	描述
CompareValidator	把一个输入控件的值与另一个输入控件的值或一个固定的值进行对比
CustomValidator	允许您编写一个方法，来处理输入值的验证
RangeValidator	检查用户输入值是否介于两个值之间
RegularExpressionValidator	确保输入控件的值匹配指定的模式
RequiredFieldValidator	使输入控件成为必需（必填）的字段
ValidationSummary	显示网页中所有验证错误的报告

W3School C# 教程

来源：C#教程

整理：飞龙

C# 基础

C# 简介

C# 是一个现代的、通用的、面向对象的编程语言，它是由微软（Microsoft）开发的，由 Ecma 和 ISO 核准认可的。

C# 是由 Anders Hejlsberg 和他的团队在 .Net 框架开发期间开发的。

C# 是专为公共语言基础结构（CLI）设计的。CLI 由可执行代码和运行时环境组成，允许在不同的计算机平台和体系结构上使用各种高级语言。

下面列出了 C# 成为一种广泛应用的专业语言的原因：

- 现代的、通用的编程语言。
- 面向对象。
- 面向组件。
- 容易学习。
- 结构化语言。
- 它产生高效率的程序。
- 它可以在多种计算机平台上编译。
- .Net 框架的一部分。

C# 强大的编程功能

虽然 C# 的构想十分接近于传统高级语言 C 和 C++，是一门面向对象的编程语言，但是它与 Java 非常相似，有许多强大的编程功能，因此得到广大程序员的亲睐。

下面列出 C# 一些重要的功能：

- 布尔条件（Boolean Conditions）
- 自动垃圾回收（Automatic Garbage Collection）
- 标准库（Standard Library）
- 组件版本（Assembly Versioning）
- 属性（Properties）和事件（Events）
- 委托（Delegates）和事件管理（Events Management）
- 易于使用的泛型（Generics）
- 索引器（Indexers）
- 条件编译（Conditional Compilation）
- 简单的多线程（Multithreading）
- LINQ 和 Lambda 表达式
- 集成 Windows

C# 环境

在这一章中，我们将讨论创建 C# 编程所需的工具。我们已经提到 C# 是 .Net 框架的一部分，且用于编写 .Net 应用程序。因此，在讨论运行 C# 程序的可用工具之前，让我们先了解一下 C# 与 .Net 框架之间的关系。

.Net 框架（.Net Framework）

.Net 框架是一个创新的平台，能帮您编写出下面类型的应用程序：

- Windows 应用程序
- Web 应用程序
- Web 服务

.Net 框架应用程序是多平台的应用程序。框架的设计方式使它适用于下列各种语言：C#、C++、Visual Basic、Jscript、COBOL 等等。所有这些语言可以访问框架，彼此之间也可以互相交互。

.Net 框架由一个巨大的代码库组成，用于 C# 等客户端语言。下面列出一些 .Net 框架的组件：

- 公共语言运行库（Common Language Runtime - CLR）
- .Net 框架类库（.Net Framework Class Library）
- 公共语言规范（Common Language Specification）
- 通用类型系统（Common Type System）
- 元数据（Metadata）和组件（Assemblies）
- Windows 窗体（Windows Forms）
- ASP.Net 和 ASP.Net AJAX
- ADO.Net
- Windows 工作流基础（Windows Workflow Foundation - WF）
- Windows 显示基础（Windows Presentation Foundation）
- Windows 通信基础（Windows Communication Foundation - WCF）
- LINQ

如需了解每个组件的详细信息，请参阅微软（Microsoft）的文档。

C# 的集成开发环境（Integrated Development Environment - IDE）

微软（Microsoft）提供了下列用于 C# 编程的开发工具：

- Visual Studio 2010 (VS)
- Visual C# 2010 Express (VCE)
- Visual Web Developer

后面两个是免费使用的，可从微软官方网址下载。使用这些工具，您可以编写各种 C# 程序，从简单的命令行应用程序到更复杂的应用程序。您也可以使用基本的文本编辑器（比如 Notepad）编写 C# 源代码文件，并使用命令行编译器（.NET 框架的一部分）编译代码为组件。

Visual C# Express 和 Visual Web Developer Express 版本是 Visual Studio 的定制版本，且具有相同的外观和感观。它们保留 Visual Studio 的大部分功能。在本教程中，我们使用的是 Visual C# 2010 Express。

您可以从 Microsoft Visual Studio 上进行下载。它会自动安装在您的机器上。请注意，您需要一个可用的网络连接来完成速成版的安装。

在 Linux 或 Mac OS 上编写 C# 程序

虽然 .NET 框架是运行在 Windows 操作系统上，但是也有一些运行于其它操作系统上的版本可供选择。**Mono** 是 .NET 框架的一个开源版本，它包含了一个 C# 编译器，且可运行于多种操作系统上，比如各种版本的 Linux 和 Mac OS。如需了解更多详情，请访问 Go Mono。

Mono 的目的不仅仅是跨平台地运行微软 .NET 应用程序，而且也为 Linux 开发者提供了更好的开发工具。Mono 可运行在多种操作系统上，包括 Android、BSD、iOS、Linux、OS X、Windows、Solaris 和 UNIX。

C# 程序结构

在我们学习 C# 编程语言的基础构件块之前，让我们先看一下 C# 的最小的程序结构，以便作为接下来章节的参考。

C# Hello World 实例

一个 C# 程序主要包括以下部分：

- 命名空间声明（Namespace declaration）
- 一个 class
- Class 方法
- Class 属性
- 一个 Main 方法
- 语句（Statements）& 表达式（Expressions）
- 注释

让我们看一个可以打印出 "Hello World" 的简单的代码：

```
using System;
namespace HelloWorldApplication
{
   class HelloWorld
   {
      static void Main(string[] args)
      {
         /* 我的第一个 C# 程序*/
         Console.WriteLine("Hello World");
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Hello World
```

让我们看一下上面程序的各个部分：

- 程序的第一行 **using System; - using** 关键字用于在程序中包含 **System** 命名空间。 一个程序一般有多个 **using** 语句。
- 下一行是 **namespace** 声明。一个 **namespace** 是一系列的类。*HelloWorldApplication* 命名空间包含了类 *HelloWorld*。
- 下一行是 **class** 声明。类 *HelloWorld* 包含了程序使用的数据和方法声明。类一般包含多个方法。方法定义了类的行为。在这里，*HelloWorld* 类只有一个 **Main** 方法。

- 下一行定义了 **Main** 方法，是所有 C# 程序的 入口点。**Main** 方法说明当执行时 类将做什么动作。
- 下一行 /.../ 将会被编译器忽略，且它会在程序中添加额外的 注释。
- Main 方法通过语句 **Console.WriteLine("Hello World");** 指定了它的行为。

  WriteLine 是一个定义在 *System* 命名空间中的 *Console* 类的一个方法。该语句会在屏幕上显示消息 "Hello, World!"。

- 最后一行 **Console.ReadKey();** 是针对 VS.NET 用户的。这使得程序会等待一个按键的动作，防止程序从 Visual Studio .NET 启动时屏幕会快速运行并关闭。

以下几点值得注意：

- C# 是大小写敏感的。
- 所有的语句和表达式必须以分号（;）结尾。
- 程序的执行从 Main 方法开始。
- 与 Java 不同的是，文件名可以不同于类的名称。

编译 & 执行 C# 程序

如果您使用 Visual Studio.Net 编译和执行 C# 程序，请按下面的步骤进行：

- 启动 Visual Studio。
- 在菜单栏上，选择 File -> New -> Project。
- 从模板中选择 Visual C#，然后选择 Windows。
- 选择 Console Application。
- 为您的项目制定一个名称，然后点击 OK 按钮。
- 新项目会出现在解决方案资源管理器（Solution Explorer）中。
- 在代码编辑器（Code Editor）中编写代码。
- 点击 Run 按钮或者按下 F5 键来运行程序。会出现一个命令提示符窗口（Command Prompt window），显示 Hello World。

您也可以使用命令行代替 Visual Studio IDE 来编译 C# 程序：

- 打开一个文本编辑器，添加上面提到的代码。
- 保存文件为 **helloworld.cs**。
- 打开命令提示符工具，定位到文件所保存的目录。
- 键入 **csc helloworld.cs** 并按下 enter 键来编译代码。
- 如果代码没有错误，命令提示符会进入下一行，并生成 **helloworld.exe** 可执行文件。
- 接下来，键入 **helloworld** 来执行程序。
- 您将看到 "Hello World" 打印在屏幕上。

C# 基本语法

C# 是一种面向对象的编程语言。在面向对象的程序设计方法中，程序由各种相互交互的对象组成。相同种类的对象通常具有相同的类型，或者说，是在相同的 class 中。

例如，以 Rectangle（矩形）对象为例。它具有 length 和 width 属性。根据设计，它可能需要接受这些属性值、计算面积和显示细节。

让我们来看看一个 Rectangle（矩形）类的实现，并借此讨论 C# 的基本语法：

```
using System;
namespace RectangleApplication
{
    class Rectangle
    {
        // 成员变量
        double length;
        double width;
        public void Acceptdetails()
        {
            length = 4.5;
            width = 3.5;
        }
        public double GetArea()
        {
            return length * width;
        }
        public void Display()
        {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    }

    class ExecuteRectangle
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.Acceptdetails();
            r.Display();
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Length: 4.5
Width: 3.5
Area: 15.75
```

using 关键字

在任何 C# 程序中的第一条语句都是：

```
using System;
```

using 关键字用于在程序中包含命名空间。一个程序可以包含多个 using 语句。

class 关键字

class 关键字用于声明一个类。

C# 中的注释

注释是用于解释代码。编译器会忽略注释的条目。在 C# 程序中，多行注释以 / 开始，并以字符 / 终止，如下所示：

```
/* This program demonstrates
The basic syntax of C# programming
Language */
```

单行注释是用 '//' 符号表示。例如：

```
}//end class Rectangle
```

成员变量

变量是类的属性或数据成员，用于存储数据。在上面的程序中，*Rectangle* 类有两个成员变量，名为 *length* 和 *width*。

成员函数

函数是一系列执行指定任务的语句。类的成员函数是在类内声明的。我们举例的类 Rectangle 包含了三个成员函数： *AcceptDetails*、*GetArea* 和 *Display*。

实例化一个类

在上面的程序中，类 *ExecuteRectangle* 是一个包含 *Main()* 方法和实例化 *Rectangle* 类的类。

标识符

标识符是用来识别类、变量、函数或任何其它用户定义的项目。在 C# 中，类的命名必须遵循如下基本规则：

- 标识符必须以字母开头，后面可以跟一系列的字母、数字（0 - 9）或下划线（_）。标识符中的第一个字符不能是数字。
- 标识符必须不包含任何嵌入的空格或符号，比如 ? - +! @ # % ^ & * ( ) [ ] { } . ; : " ' / \。但是，可以使用下划线（_）。
- 标识符不能是 C# 关键字。

C# 关键字

关键字是 C# 编译器预定义的保留字。这些关键字不能用作标识符，但是，如果您想使用这些关键字作为标识符，可以在关键字前面加上 @ 字符作为前缀。

在 C# 中，有些标识符在代码的上下文中有特殊的意义，如 get 和 set，这些被称为上下文关键字（contextual keywords）。

下表列出了 C# 中的保留关键字（Reserved Keywords）和上下文关键字（Contextual Keywords）：

保留关键字						
abstract	as	base	bool	break	byte	case
catch	char	checked	class	const	continue	decin
default	delegate	do	double	else	enum	event
explicit	extern	false	finally	fixed	float	for
foreach	goto	if	implicit	in	in (generic modifier)	int
interface	internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override	parar
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					
上下文关键字						
add	alias	ascending	descending	dynamic	from	get
global	group	into	join	let	orderby	partia (type
partial (method)	remove	select	set			

C# 数据类型

在 C# 中，变量分为以下几种类型：

- 值类型（Value types）
- 引用类型（Reference types）
- 指针类型（Pointer types）

值类型（Value types）

值类型变量可以直接分配给一个值。它们是从类 **System.ValueType** 中派生的。

值类型直接包含数据。比如 **int**、**char**、**float**，它们分别存储数字、字母、浮点数。当您声明一个 **int** 类型时，系统分配内存来存储值。

下表列出了 C# 2010 中可用的值类型：

类型	描述	范围	默认值
bool	布尔值	True 或 False	False
byte	8 位无符号整数	0 到 255	0
char	16 位 Unicode 字符	U +0000 到 U +ffff	'\0'
decimal	128 位精确的十进制值，28-29 有效位数	(-7.9 x 10^{28} 到 7.9 x 10^{28}) / 10^{0} 到 28	0.0M
double	64 位双精度浮点型	(+/-)5.0 x 10^{-324} 到 (+/-)1.7 x 10^{308}	0.0D
float	32 位单精度浮点型	-3.4 x 10^{38} 到 + 3.4 x 10^{38}	0.0F
int	32 位有符号整数类型	-2,147,483,648 到 2,147,483,647	0
long	64 位有符号整数类型	-923,372,036,854,775,808 到 9,223,372,036,854,775,807	0L
sbyte	8 位有符号整数类型	-128 到 127	0
short	16 位有符号整数类型	-32,768 到 32,767	0
uint	32 位无符号整数类型	0 到 4,294,967,295	0
ulong	64 位无符号整数类型	0 到 18,446,744,073,709,551,615	0
ushort	16 位无符号整数类型	0 到 65,535	0

如需得到一个类型或一个变量在特定平台上的准确尺寸，可以使用 **sizeof** 方法。表达式 *sizeof(type)* 产生以字节为单位存储对象或类型的存储尺寸。下面举例获取任何机器上 *int* 类型的存储尺寸：

```
namespace DataTypeApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Size of int: {0}", sizeof(int));
         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Size of int: 4
```

引用类型（Reference types）

引用类型不包含存储在变量中的实际数据，但它们包含对变量的引用。

换句话说，它们指的是一个内存位置。使用多个变量时，引用类型可以指向一个内存位置。如果内存位置的数据是由一个变量改变的，其他变量会自动反映这种值的变化。内置的 引用类型有：**object**、**dynamic** 和 **string**。

对象（Object）类型

对象（**Object**）类型 是 C# 通用类型系统（Common Type System - CTS）中所有数据类型的终极基类。Object 是 System.Object 类的别名。所以对象（Object）类型可以被分配任何其他类型（值类型、引用类型、预定义类型或用户自定义类型）的值。但是，在分配值之前，需要先进行类型转换。

当一个值类型转换为对象类型时，则被称为 装箱；另一方面，当一个对象类型转换为值类型时，则被称为 拆箱。

```
object obj;
obj = 100; // 这是装箱
```

动态（Dynamic）类型

您可以存储任何类型的值在动态数据类型变量中。这些变量的类型检查是在运行时发生的。

声明动态类型的语法：

```
dynamic <variable_name> = value;
```

例如：

```
dynamic d = 20;
```

动态类型与对象类型相似，但是对象类型变量的类型检查是在编译时发生的，而动态类型变量的类型检查是在运行时发生的。

字符串（String）类型

字符串（**String**）类型 允许您给变量分配任何字符串值。字符串（String）类型是 System.String 类的别名。它是从对象（Object）类型派生的。字符串（String）类型的值可以通过两种形式进行分配：引号和 @引号。

例如：

```
String str = "w3cschool.cc";
```

一个 @引号字符串：

```
@"w3cschool.cc";
```

C# string 字符串的前面可以加 @（称作"逐字字符串"）将转义字符（\）当作普通字符对待，比如：

```
string str = @"C:\Windows";
```

等价于：

```
string str = "C:\\Windows";
```

@ 字符串中可以任意换行，换行符及缩进空格都计算在字符串长度之内。

```
string str = @"<script type=""text/javascript"">
    <!--
    -->
    </script>";
```

用户自定义引用类型有：class、interface 或 delegate。我们将在以后的章节中讨论这些类型。

指针类型（Pointer types）

指针类型变量存储另一种类型的内存地址。C# 中的指针与 C 或 C++ 中的指针有相同的功能。

声明指针类型的语法：

```
type* identifier;
```

例如：

```
char* cptr;
int* iptr;
```

我们将在章节"不安全的代码"中讨论指针类型。

C# 类型转换

类型转换从根本上说是类型铸造，或者说是把数据从一种类型转换为另一种类型。在 C# 中，类型铸造有两种形式：

- 隐式类型转换 - 这些转换是 C# 默认的以安全方式进行的转换。例如，从小的整数类型转换为大的整数类型，从派生类转换为基类。
- 显式类型转换 - 这些转换是通过用户使用预定义的函数显式完成的。显式转换需要强制转换运算符。

下面的实例显示了一个显式的类型转换：

```
namespace TypeConversionApplication
{
    class ExplicitConversion
    {
        static void Main(string[] args)
        {
            double d = 5673.74;
            int i;

            // 强制转换 double 为 int
            i = (int)d;
            Console.WriteLine(i);
            Console.ReadKey();

        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
5673
```

C# 类型转换方法

C# 提供了下列内置的类型转换方法：

方法	描述
ToBoolean	如果可能的话，把类型转换为布尔型。
ToByte	把类型转换为字节类型。
ToChar	如果可能的话，把类型转换为单个 Unicode 字符类型。
ToDateTime	把类型（整数或字符串类型）转换为 日期-时间 结构。
ToDecimal	把浮点型或整数类型转换为十进制类型。
ToDouble	把类型转换为双精度浮点型。
ToInt16	把类型转换为 16 位整数类型。
ToInt32	把类型转换为 32 位整数类型。
ToInt64	把类型转换为 64 位整数类型。
ToSbyte	把类型转换为有符号字节类型。
ToSingle	把类型转换为小浮点数类型。
ToString	把类型转换为字符串类型。
ToType	把类型转换为指定类型。
ToUInt16	把类型转换为 16 位无符号整数类型。
ToUInt32	把类型转换为 32 位无符号整数类型。
ToUInt64	把类型转换为 64 位无符号整数类型。

下面的实例把不同值的类型转换为字符串类型：

```
namespace TypeConversionApplication
{
    class StringConversion
    {
        static void Main(string[] args)
        {
            int i = 75;
            float f = 53.005f;
            double d = 2345.7652;
            bool b = true;

            Console.WriteLine(i.ToString());
            Console.WriteLine(f.ToString());
            Console.WriteLine(d.ToString());
            Console.WriteLine(b.ToString());
            Console.ReadKey();

        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
75
53.005
2345.7652
True
```

C# 变量

一个变量只不过是一个供程序操作的存储区的名字。在 C# 中，每个变量都有一个特定的类型，类型决定了变量的内存大小和布局。范围内的值可以存储在内存中，可以对变量进行一系列操作。

我们已经讨论了各种数据类型。C# 中提供的基本的值类型大致可以分为以下几类：

类型	举例
整数类型	sbyte、byte、short、ushort、int、uint、long、ulong 和 char
浮点型	float 和 double
十进制类型	decimal
布尔类型	true 或 false 值，指定的值
空类型	可为空值的数据类型

C# 允许定义其他值类型的变量，比如 **enum**，也允许定义引用类型变量，比如 **class**。这些我们将在以后的章节中进行讨论。在本章节中，我们只研究基本变量类型。

C# 中的变量定义

C# 中变量定义的语法：

```
<data_type> <variable_list>;
```

在这里，data_type 必须是一个有效的 C# 数据类型，可以是 char、int、float、double 或其他用户自定义的数据类型。variable_list 可以由一个或多个用逗号分隔的标识符名称组成。

一些有效的变量定义如下所示：

```
int i, j, k;
char c, ch;
float f, salary;
double d;
```

您可以在变量定义时进行初始化：

```
int i = 100;
```

C# 中的变量初始化

变量通过在等号后跟一个常量表达式进行初始化（赋值）。初始化的一般形式为：

```
variable_name = value;
```

变量可以在声明时被初始化（指定一个初始值）。初始化由一个等号后跟一个常量表达式组成，如下所示：

```
<data_type> <variable_name> = value;
```

一些实例：

```
int d = 3, f = 5;    /* 初始化 d 和 f. */
byte z = 22;         /* 初始化 z. */
double pi = 3.14159; /* 声明 pi 的近似值 */
char x = 'x';        /* 变量 x 的值为 'x' */
```

正确地初始化变量是一个良好的编程习惯，否则有时程序会产生意想不到的结果。

请看下面的实例，使用了各种类型的变量：

```
namespace VariableDefinition
{
    class Program
    {
        static void Main(string[] args)
        {
            short a;
            int b ;
            double c;

            /* 实际初始化 */
            a = 10;
            b = 20;
            c = a + b;
            Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
a = 10, b = 20, c = 30
```

接受来自用户的值

System 命名空间中的 **Console** 类提供了一个函数 **ReadLine()**，用于接收来自用户的输入，并把它存储到一个变量中。

例如：

```
int num;
num = Convert.ToInt32(Console.ReadLine());
```

函数 **Convert.ToInt32()** 把用户输入的数据转换为 int 数据类型，因为 **Console.ReadLine()** 只接受字符串格式的数据。

C# 中的 Lvalues 和 Rvalues

C# 中的两种表达式：

1. **lvalue**：lvalue 表达式可以出现在赋值语句的左边或右边。
2. **rvalue**：rvalue 表达式可以出现在赋值语句的右边，不能出现在赋值语句的左边。

变量是 lvalue 的，所以可以出现在赋值语句的左边。数值是 rvalue 的，因此不能被赋值，不能出现在赋值语句的左边。下面是一个有效的语句：

```
int g = 20;
```

下面是一个无效的语句，会产生编译时错误：

```
10 = 20;
```

C# 常量

常量是固定值，程序执行期间不会改变。常量可以是任何基本数据类型，比如整数常量、浮点常量、字符常量或者字符串常量，还有枚举常量。

常量可以被当作常规的变量，只是它们的值在定义后不能被修改。

整数常量

整数常量可以是十进制、八进制或十六进制的常量。前缀指定基数：0x 或 0X 表示十六进制，0 表示八进制，没有前缀则表示十进制。

整数常量也可以有后缀，可以是 U 和 L 的组合，其中，U 和 L 分别表示 unsigned 和 long。后缀可以是大写或者小写，多个后缀以任意顺序进行组合。

这里有一些整数常量的实例：

```
212         /* 合法 */
215u        /* 合法 */
0xFeeL      /* 合法 */
078         /* 非法：8 不是一个八进制数字 */
032UU       /* 非法：不能重复后缀 */
```

以下是各种类型的整数常量的实例：

```
85         /* 十进制 */
0213       /* 八进制 */
0x4b       /* 十六进制 */
30         /* int */
30u        /* 无符号 int */
30l        /* long */
30ul       /* 无符号 long */
```

浮点常量

一个浮点常量是由整数部分、小数点、小数部分和指数部分组成。您可以使用小数形式或者指数形式来表示浮点常量。

这里有一些浮点常量的实例：

```
3.14159       /* 合法 */
314159E-5L    /* 合法 */
510E          /* 非法：不完全指数 */
210f          /* 非法：没有小数或指数 */
.e55          /* 非法：缺少整数或小数 */
```

使用小数形式表示时，必须包含小数点、指数或同时包含两者。使用指数形式表示时，必须包含整数部分、小数部分或同时包含两者。有符号的指数是用 e 或 E 表示的。

字符常量

字符常量是括在单引号里，例如，'x'，且可存储在一个简单的字符类型变量中。一个字符常量可以是一个普通字符（例如 'x'）、一个转义序列（例如 '\t'）或者一个通用字符（例如 '\u02C0'）。

在 C# 中有一些特定的字符，当它们的前面带有反斜杠时有特殊的意义，可用于表示换行符（\n）或制表符 tab（\t）。在这里，列出一些转义序列码：

转义序列	含义
\	\ 字符
\'	' 字符
\"	" 字符
\?	? 字符
\a	Alert 或 bell
\b	退格键（Backspace）
\f	换页符（Form feed）
\n	换行符（Newline）
\r	回车
\t	水平制表符 tab
\v	垂直制表符 tab
\ooo	一到三位的八进制数
\xhh . . .	一个或多个数字的十六进制数

以下是一些转义序列字符的实例：

```
namespace EscapeChar
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello\tWorld\n\n");
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Hello   World
```

字符串常量

字符常量是括在双引号 "" 里，或者是括在 @"" 里。字符串常量包含的字符与字符常量相似，可以是：普通字符、转义序列和通用字符

使用字符串常量时，可以把一个很长的行拆成多个行，可以使用空格分隔各个部分。

这里是一些字符串常量的实例。下面所列的各种形式表示相同的字符串。

```
"hello, dear"
"hello, \
dear"
"hello, " "d" "ear"
@"hello dear"
```

定义常量

常量是使用 **const** 关键字来定义的 。定义一个常量的语法如下：

```
const <data_type> <constant_name> = value;
```

下面的代码演示了如何在程序中定义和使用常量：

```
using System;

namespace DeclaringConstants
{
    class Program
    {
        static void Main(string[] args)
        {
            const double pi = 3.14159; // 常量声明
            double r;
            Console.WriteLine("Enter Radius: ");
            r = Convert.ToDouble(Console.ReadLine());
            double areaCircle = pi * r * r;
            Console.WriteLine("Radius: {0}, Area: {1}", r, areaCircle);
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Enter Radius:
3
Radius: 3, Area: 28.27431
```

C# 运算符

运算符是一种告诉编译器执行特定的数学或逻辑操作的符号。C# 有丰富的内置运算符，分类如下：

- 算术运算符
- 关系运算符
- 逻辑运算符
- 位运算符
- 赋值运算符
- 杂项运算符

本教程将逐一讲解算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符及其他运算符。

算术运算符

下表显示了 C# 支持的所有算术运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

运算符	描述	实例
+	把两个操作数相加	A + B 将得到 30
-	从第一个操作数中减去第二个操作数	A - B 将得到 -10
*	把两个操作数相乘	A * B 将得到 200
/	分子除以分母	B / A 将得到 2
%	取模运算符，整除后的余数	B % A 将得到 0
++	自增运算符，整数值增加 1	A++ 将得到 11
--	自减运算符，整数值减少 1	A-- 将得到 9

实例

请看下面的实例，了解 C# 中所有可用的算术运算符：

```
using System;

namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 21;
            int b = 10;
            int c;

            c = a + b;
            Console.WriteLine("Line 1 - c 的值是 {0}", c);
            c = a - b;
            Console.WriteLine("Line 2 - c 的值是 {0}", c);
            c = a * b;
            Console.WriteLine("Line 3 - c 的值是 {0}", c);
            c = a / b;
            Console.WriteLine("Line 4 - c 的值是 {0}", c);
            c = a % b;
            Console.WriteLine("Line 5 - c 的值是 {0}", c);
            c = a++;
            Console.WriteLine("Line 6 - c 的值是 {0}", c);
            c = a--;
            Console.WriteLine("Line 7 - c 的值是 {0}", c);
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Line 1 - c 的值是 31
Line 2 - c 的值是 11
Line 3 - c 的值是 210
Line 4 - c 的值是 2
Line 5 - c 的值是 1
Line 6 - c 的值是 21
Line 7 - c 的值是 22
```

关系运算符

下表显示了 C# 支持的所有关系运算符。假设变量 **A** 的值为 10，变量 **B** 的值为 20，则：

运算符	描述	实例
==	检查两个操作数的值是否相等，如果相等则条件为真。	(A == B) 不为真。
!=	检查两个操作数的值是否相等，如果不相等则条件为真。	(A != B) 为真。
>	检查左操作数的值是否大于右操作数的值，如果是则条件为真。	(A > B) 不为真。
<	检查左操作数的值是否小于右操作数的值，如果是则条件为真。	(A < B) 为真。
>=	检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。	(A >= B) 不为真。
<=	检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。	(A <= B) 为真。

实例

请看下面的实例，了解 C# 中所有可用的关系运算符：

```
using System;

class Program
{
  static void Main(string[] args)
  {
      int a = 21;
      int b = 10;

      if (a == b)
      {
          Console.WriteLine("Line 1 - a 等于 b");
      }
      else
      {
          Console.WriteLine("Line 1 - a 不等于 b");
      }
      if (a < b)
      {
          Console.WriteLine("Line 2 - a 小于 b");
      }
      else
      {
          Console.WriteLine("Line 2 - a 不小于 b");
      }
      if (a > b)
      {
          Console.WriteLine("Line 3 - a 大于 b");
      }
      else
      {
          Console.WriteLine("Line 3 - a 不大于 b");
      }
      /* 改变 a 和 b 的值 */
      a = 5;
      b = 20;
      if (a <= b)
      {
         Console.WriteLine("Line 4 - a 小于或等于 b");
      }
      if (b >= a)
      {
         Console.WriteLine("Line 5 - b 大于或等于 a");
      }
  }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Line 1 - a 不等于 b
Line 2 - a 不小于 b
Line 3 - a 大于 b
Line 4 - a 小于或等于 b
Line 5 - b 大于或等于 a
```

逻辑运算符

下表显示了 C# 支持的所有逻辑运算符。假设变量 **A** 为布尔值 true，变量 **B** 为布尔值 false，则：

运算符	描述	实例
&&	称为逻辑与运算符。如果两个操作数都非零，则条件为真。	(A && B) 为假。
|:|:	称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。	(A |:|: B) 为真。
!	称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。	!(A && B) 为真。

实例

请看下面的实例，了解 C# 中所有可用的逻辑运算符：

```
using System;

namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            bool a = true;
            bool b = true;

            if (a && b)
            {
               Console.WriteLine("Line 1 - 条件为真");
            }
            if (a || b)
            {
                Console.WriteLine("Line 2 - 条件为真");
            }
            /* 改变 a 和 b 的值 */
            a = false;
            b = true;
            if (a && b)
            {
                Console.WriteLine("Line 3 - 条件为真");
            }
            else
            {
                Console.WriteLine("Line 3 - 条件不为真");
            }
            if (!(a && b))
            {
                Console.WriteLine("Line 4 - 条件为真");
            }
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Line 1 - 条件为真
Line 2 - 条件为真
Line 3 - 条件不为真
Line 4 - 条件为真
```

位运算符

位运算符作用于位，并逐位执行操作。&、 | 和 ^ 的真值表如下所示：

<table>
<tr><th>p</th><th>q</th><th>p & q</th><th>p | q</th><th>p ^ q</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table>

假设如果 A = 60，且 B = 13，现在以二进制格式表示，它们如下所示：

```
A = 0011 1100

B = 0000 1101

-----------------

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011
```

下表列出了 C# 支持的位运算符。假设变量 **A** 的值为 60，变量 **B** 的值为 13，则：

<table>
<tr><th>运算符</th><th>描述</th><th>实例</th></tr>
<tr><td>&</td><td>如果同时存在于两个操作数中，二进制 AND 运算符复制一位到结果中。</td><td>(A & B) 将得到 12，即为 0000 1100</td></tr>
<tr><td>|</td><td>如果存在于任一操作数中，二进制 OR 运算符复制一位到结果中。</td><td>(A | B) 将得到 61，即为 0011 1101</td></tr>
<tr><td>^</td><td>如果存在于其中一个操作数中但不同时存在于两个操作数中，二进制异或运算符复制一位到结果中。</td><td>(A ^ B) 将得到 49，即为 0011 0001</td></tr>
<tr><td>~</td><td>二进制补码运算符是一元运算符，具有"翻转"位效果。</td><td>(~A) 将得到 -61，即为 1100 0011，2 的补码形式，带符号的二进制数。</td></tr>
<tr><td><<</td><td>二进制左移运算符。左操作数的值向左移动右操作数指定的位数。</td><td>A << 2 将得到 240，即为 1111 0000</td></tr>
<tr><td>>></td><td>二进制右移运算符。左操作数的值向右移动右操作数指定的位数。</td><td>A >> 2 将得到 15，即为 0000 1111</td></tr>
</table>

实例

请看下面的实例，了解 C# 中所有可用的位运算符：

```
using System;
namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 60;               /* 60 = 0011 1100 */
            int b = 13;               /* 13 = 0000 1101 */
            int c = 0;

             c = a & b;           /* 12 = 0000 1100 */
             Console.WriteLine("Line 1 - c 的值是 {0}", c );

             c = a | b;           /* 61 = 0011 1101 */
             Console.WriteLine("Line 2 - c 的值是 {0}", c);

             c = a ^ b;           /* 49 = 0011 0001 */
             Console.WriteLine("Line 3 - c 的值是 {0}", c);

             c = ~a;               /*-61 = 1100 0011 */
             Console.WriteLine("Line 4 - c 的值是 {0}", c);

             c = a << 2;     /* 240 = 1111 0000 */
             Console.WriteLine("Line 5 - c 的值是 {0}", c);

             c = a >> 2;     /* 15 = 0000 1111 */
             Console.WriteLine("Line 6 - c 的值是 {0}", c);
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Line 1 - c 的值是 12
Line 2 - c 的值是 61
Line 3 - c 的值是 49
Line 4 - c 的值是 -61
Line 5 - c 的值是 240
Line 6 - c 的值是 15
```

赋值运算符

下表列出了 C# 支持的赋值运算符：

运算符	描述	实例
=	简单的赋值运算符，把右边操作数的值赋给左边操作数	C = A + B 将把 A + B 的值赋给 C
+=	加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数	C += A 相当于 C = C + A
-=	减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数	C -= A 相当于 C = C - A
*=	乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数	C = *A* 相当于 *C* = *C* A
/=	除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数	C /= A 相当于 C = C / A
%=	求模且赋值运算符，求两个操作数的模赋值给左边操作数	C %= A 相当于 C = C % A
<<=	左移且赋值运算符	C <<= 2 等同于 C = C << 2
>>=	右移且赋值运算符	C >>= 2 等同于 C = C >> 2
&=	按位与且赋值运算符	C &= 2 等同于 C = C & 2
^=	按位异或且赋值运算符	C ^= 2 等同于 C = C ^ 2
|:=	按位或且赋值运算符	C |:= 2 等同于 C = C |: 2

实例

请看下面的实例，了解 C# 中所有可用的赋值运算符：

```
using System;

namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 21;
            int c;

            c = a;
            Console.WriteLine("Line 1 - =  c 的值 = {0}", c);

            c += a;
            Console.WriteLine("Line 2 - += c 的值 = {0}", c);

            c -= a;
            Console.WriteLine("Line 3 - -=  c 的值 = {0}", c);

            c *= a;
            Console.WriteLine("Line 4 - *=  c 的值 = {0}", c);

            c /= a;
            Console.WriteLine("Line 5 - /=  c 的值 = {0}", c);

            c = 200;
            c %= a;
            Console.WriteLine("Line 6 - %=  c 的值 = {0}", c);

            c <<= 2;
            Console.WriteLine("Line 7 - <<=  c 的值 = {0}", c);

            c >>= 2;
            Console.WriteLine("Line 8 - >>=  c 的值 = {0}", c);

            c &= 2;
            Console.WriteLine("Line 9 - &=  c 的值 = {0}", c);

            c ^= 2;
            Console.WriteLine("Line 10 - ^=  c 的值 = {0}", c);

            c |= 2;
            Console.WriteLine("Line 11 - |=  c 的值 = {0}", c);
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Line 1 - =     c 的值 = 21
Line 2 - +=    c 的值 = 42
Line 3 - -=    c 的值 = 21
Line 4 - *=    c 的值 = 441
Line 5 - /=    c 的值 = 21
Line 6 - %=    c 的值 = 11
Line 7 - <<=    c 的值 = 44
Line 8 - >>=    c 的值 = 11
Line 9 - &=    c 的值 = 2
Line 10 - ^=    c 的值 = 0
Line 11 - |=    c 的值 = 2
```

杂项运算符

下表列出了 C# 支持的其他一些重要的运算符，包括 **sizeof**、**typeof** 和 **? :**。

运算符	描述	实例
sizeof()	返回数据类型的大小。	sizeof(int)，将返回 4.
typeof()	返回 class 的类型。	typeof(StreamReader);
&	返回变量的地址。	&a; 将得到变量的实际地址。
*	变量的指针。	*a; 将指向一个变量。
? :	条件表达式	如果条件为真 ? 则为 X : 否则为 Y
is	判断对象是否为某一类型。	If(Ford is Car) // 检查 Ford 是否是 Car 类的一个对象。
as	强制转换，即使转换失败也不会抛出异常。	Object obj = new StringReader("Hello"); StringReader r = obj as StringReader;

实例

```
using System;

namespace OperatorsAppl
{

   class Program
   {
      static void Main(string[] args)
      {

         /* sizeof 运算符的实例 */
         Console.WriteLine("int 的大小是 {0}", sizeof(int));
         Console.WriteLine("short 的大小是 {0}", sizeof(short));
         Console.WriteLine("double 的大小是 {0}", sizeof(double));

         /* 三元运算符符的实例 */
         int a, b;
         a = 10;
         b = (a == 1) ? 20 : 30;
         Console.WriteLine("b 的值是 {0}", b);

         b = (a == 10) ? 20 : 30;
         Console.WriteLine("b 的值是 {0}", b);
         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
int 的大小是 4
short 的大小是 2
double 的大小是 8
b 的值是 30
b 的值是 20
```

C# 中的运算符优先级

运算符的优先级确定表达式中项的组合。这会影响到一个表达式如何计算。某些运算符比其他运算符有更高的优先级，例如，乘除运算符具有比加减运算符更高的优先级。

例如 x = 7 + 3 *2*，在这里，*x* 被赋值为 *13*，而不是 *20*，因为运算符 具有比 + 更高的优先级，所以首先计算乘法 3*2，然后再加上 7。

下表将按运算符优先级从高到低列出各个运算符，具有较高优先级的运算符出现在表格的上面，具有较低优先级的运算符出现在表格的下面。在表达式中，较高优先级的运算符会优先被计算。

类别	运算符	结合性
后缀	() [] -> . ++ - -	从左到右
一元	+ - ! ~ ++ - - (type)* & sizeof	从右到左
乘除	* / %	从左到右
加减	+ -	从左到右
移位	<< >>	从左到右
关系	< <= > >=	从左到右
相等	== !=	从左到右
位与 AND	&	从左到右
位异或 XOR	^	从左到右
位或 OR	|	从左到右
逻辑与 AND	&&	从左到右
逻辑或 OR	||	从左到右
条件	?:	从右到左
赋值	= += -= *= /= %=>>= <<= &= ^= |=	从右到左
逗号	,	从左到右

实例

```
using System;

namespace OperatorsAppl
{

   class Program
   {
      static void Main(string[] args)
      {
         int a = 20;
         int b = 10;
         int c = 15;
         int d = 5;
         int e;
         e = (a + b) * c / d;     // ( 30 * 15 ) / 5
         Console.WriteLine("(a + b) * c / d 的值是 {0}", e);

         e = ((a + b) * c) / d;   // (30 * 15 ) / 5
         Console.WriteLine("((a + b) * c) / d 的值是 {0}", e);

         e = (a + b) * (c / d);   // (30) * (15/5)
         Console.WriteLine("(a + b) * (c / d) 的值是 {0}", e);

         e = a + (b * c) / d;    //  20 + (150/5)
         Console.WriteLine("a + (b * c) / d 的值是 {0}", e);
         Console.ReadLine();
      }
   }
}
```

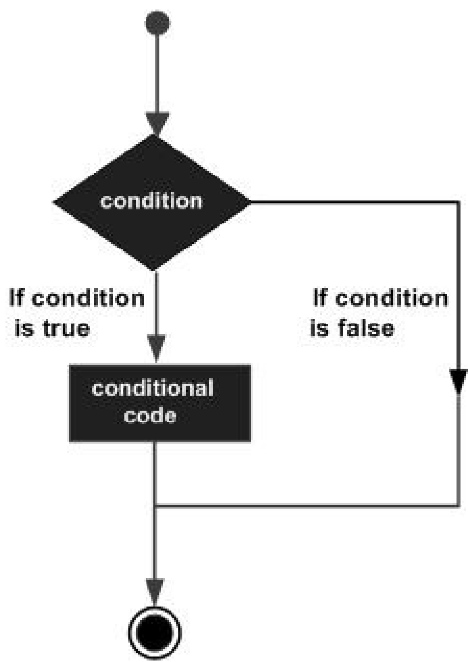
当上面的代码被编译和执行时，它会产生下列结果：

```
(a + b) * c / d 的值是 90
((a + b) * c) / d 的值是 90
(a + b) * (c / d) 的值是 90
a + (b * c) / d 的值是 50
```

C# 判断

判断结构要求程序员指定一个或多个要评估或测试的条件，以及条件为真时要执行的语句（必需的）和条件为假时要执行的语句（可选的）。

下面是大多数编程语言中典型的判断结构的一般形式：



判断语句

C# 提供了以下类型的判断语句。点击链接查看每个语句的细节。

语句	描述
if 语句	一个 **if 语句** 由一个布尔表达式后跟一个或多个语句组成。
if...else 语句	一个 **if 语句** 后可跟一个可选的 **else 语句**，else 语句在布尔表达式为假时执行。
嵌套 if 语句	您可以在一个 **if** 或 **else if** 语句内使用另一个 **if** 或 **else if** 语句。
switch 语句	一个 **switch** 语句允许测试一个变量等于多个值时的情况。
嵌套 switch 语句	您可以在一个 **switch** 语句内使用另一个 **switch** 语句。

? : 运算符

我们已经在前面的章节中讲解了 条件运算符 **? :**，可以用来替代 **if...else** 语句。它的一般形式如下：

```
Exp1 ? Exp2 : Exp3;
```

其中，Exp1、Exp2 和 Exp3 是表达式。请注意，冒号的使用和位置。

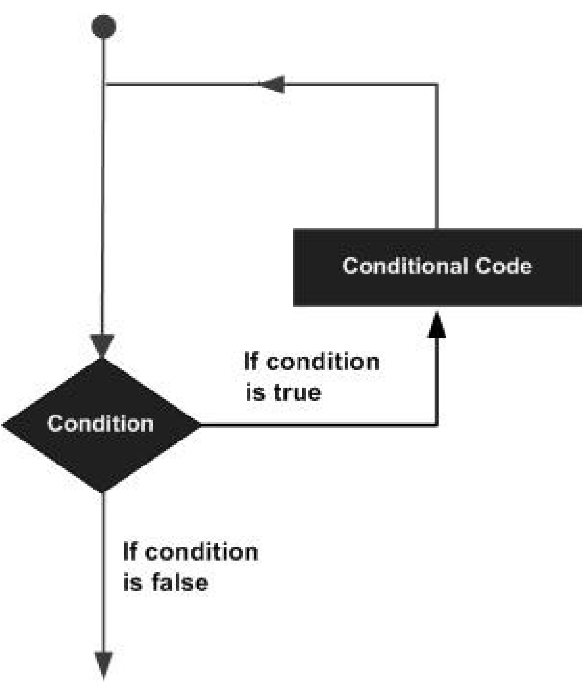
? 表达式的值是由 Exp1 决定的。如果 Exp1 为真，则计算 Exp2 的值，结果即为整个 ? 表达式的值。如果 Exp1 为假，则计算 Exp3 的值，结果即为整个 ? 表达式的值。

C# 循环

有的时候，可能需要多次执行同一块代码。一般情况下，语句是顺序执行的：函数中的第一个语句先执行，接着是第二个语句，依此类推。

编程语言提供了允许更为复杂的执行路径的多种控制结构。

循环语句允许我们多次执行一个语句或语句组，下面是大多数编程语言中循环语句的一般形式：



循环类型

C# 提供了以下几种循环类型。点击链接查看每个类型的细节。

循环类型	描述
while 循环	当给定条件为真时，重复语句或语句组。它会在执行循环主体之前测试条件。
for 循环	多次执行一个语句序列，简化管理循环变量的代码。
do...while 循环	除了它是在循环主体结尾测试条件外，其他与 while 语句类似。
嵌套循环	您可以在 while、for 或 do..while 循环内使用一个或多个循环。

循环控制语句

循环控制语句更改执行的正常序列。当执行离开一个范围时，所有在该范围中创建的自动对象都会被销毁。

C# 提供了下列的控制语句。点击链接查看每个语句的细节。

控制语句	描述
break 语句	终止 **loop** 或 **switch** 语句，程序流将继续执行紧接着 loop 或 switch 的下一条语句。
continue 语句	引起循环跳过主体的剩余部分，立即重新开始测试条件。

无限循环

如果条件永远不为假，则循环将变成无限循环。**for** 循环在传统意义上可用于实现无限循环。由于构成循环的三个表达式中任何一个都不是必需的，您可以将某些条件表达式留空来构成一个无限循环。

```
using System;

namespace Loops
{

    class Program
    {
        static void Main(string[] args)
        {
            for (; ; )
            {
                Console.WriteLine("Hey! I am Trapped");
            }

        }
    }
}
```

当条件表达式不存在时，它被假设为真。您也可以设置一个初始值和增量表达式，但是一般情况下，程序员偏向于使用 for(;;) 结构来表示一个无限循环。

C# 封装

封装 被定义为"把一个或多个项目封闭在一个物理的或者逻辑的包中"。在面向对象程序设计方法论中，封装是为了防止对实现细节的访问。

抽象和封装是面向对象程序设计的相关特性。抽象允许相关信息可视化，封装则使程序员实现所需级别的抽象。

封装使用 访问修饰符 来实现。一个 访问修饰符 定义了一个类成员的范围和可见性。C# 支持的访问修饰符如下所示：

- Public
- Private
- Protected
- Internal
- Protected internal

Public 访问修饰符

Public 访问修饰符允许一个类将其成员变量和成员函数暴露给其他的函数和对象。任何公有成员可以被外部的类访问。

下面的实例说明了这点：

```
using System;

namespace RectangleApplication
{
    class Rectangle
    {
        //成员变量
        public double length;
        public double width;

        public double GetArea()
        {
            return length * width;
        }
        public void Display()
        {
            Console.WriteLine("长度： {0}", length);
            Console.WriteLine("宽度： {0}", width);
            Console.WriteLine("面积： {0}", GetArea());
        }
    }//end class Rectangle
    class ExecuteRectangle
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.length = 4.5;
            r.width = 3.5;
            r.Display();
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
长度： 4.5
宽度： 3.5
面积： 15.75
```

在上面的实例中，成员变量 length 和 width 被声明为 **public**，所以它们可以被函数 Main() 使用 Rectangle 类的实例 **r** 访问。

成员函数 *Display()* 和 *GetArea()* 也可以不通过类的实例直接访问这些变量。

成员函数 *Display()* 也被声明为 **public**，所以它也能被 *Main()* 使用 Rectangle 类的实例 **r** 访问。

Private 访问修饰符

Private 访问修饰符允许一个类将其成员变量和成员函数对其他的函数和对象进行隐藏。只有同一个类中的函数可以访问它的私有成员。即使是类的实例也不能访问它的私有成员。

下面的实例说明了这点：

```
using System;

namespace RectangleApplication
{
    class Rectangle
    {
        //成员变量
        private double length;
        private double width;

        public void Acceptdetails()
        {
            Console.WriteLine("请输入长度：");
            length = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine("请输入宽度：");
            width = Convert.ToDouble(Console.ReadLine());
        }
        public double GetArea()
        {
            return length * width;
        }
        public void Display()
        {
            Console.WriteLine("长度： {0}", length);
            Console.WriteLine("宽度： {0}", width);
            Console.WriteLine("面积： {0}", GetArea());
        }
    }//end class Rectangle
    class ExecuteRectangle
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.Acceptdetails();
            r.Display();
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
请输入长度：
4.4
请输入宽度：
3.3
长度： 4.4
宽度： 3.3
面积： 14.52
```

在上面的实例中，成员变量 length 和 width 被声明为 **private**，所以它们不能被函数 Main() 访问。

成员函数 *AcceptDetails()* 和 *Display()* 可以访问这些变量。

由于成员函数 *AcceptDetails()* 和 *Display()* 被声明为 **public**，所以它们可以被 *Main()* 使用 Rectangle 类的实例 **r** 访问。

Protected 访问修饰符

Protected 访问修饰符允许子类访问它的基类的成员变量和成员函数。这样有助于实现继承。我们将在继承的章节详细讨论这个。更详细地讨论这个。

Internal 访问修饰符

Internal 访问说明符允许一个类将其成员变量和成员函数暴露给当前程序中的其他函数和对象。换句话说，带有 internal 访问修饰符的任何成员可以被定义在该成员所定义的应用程序内的任何类或方法访问。

下面的实例说明了这点：

```
using System;

namespace RectangleApplication
{
    class Rectangle
    {
        //成员变量
        internal double length;
        internal double width;

        double GetArea()
        {
            return length * width;
        }
       public void Display()
        {
            Console.WriteLine("长度： {0}", length);
            Console.WriteLine("宽度： {0}", width);
            Console.WriteLine("面积： {0}", GetArea());
        }
    }//end class Rectangle
    class ExecuteRectangle
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.length = 4.5;
            r.width = 3.5;
            r.Display();
            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
长度： 4.5
宽度： 3.5
面积： 15.75
```

在上面的实例中，请注意成员函数 *GetArea()* 声明的时候不带有任何访问修饰符。如果没有指定访问修饰符，则使用类成员的默认访问修饰符，即为 **private**。

Protected Internal 访问修饰符

Protected Internal 访问修饰符允许一个类将其成员变量和成员函数对同一应用程序内的子类以外的其他的类对象和函数进行隐藏。这也被用于实现继承。

C# 方法

一个方法是把一些相关的语句组织在一起，用来执行一个任务的语句块。每一个 C# 程序至少有一个带有 Main 方法的类。

要使用一个方法，您需要：

- 定义方法
- 调用方法

C# 中定义方法

当定义一个方法时，从根本上说是在声明它的结构的元素。在 C# 中，定义方法的语法如下：

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
   Method Body
}
```

下面是方法的各个元素：

- **Access Specifier**：访问修饰符，这个决定了变量或方法对于另一个类的可见性。
- **Return type**：返回类型，一个方法可以返回一个值。返回类型是方法返回的值的数据类型。如果方法不返回任何值，则返回类型为 **void**。
- **Method name**：方法名称，是一个唯一的标识符，且是大小写敏感的。它不能与类中声明的其他标识符相同。
- **Parameter list**：参数列表，使用圆括号括起来，该参数是用来传递和接收方法的数据。参数列表是指方法的参数类型、顺序和数量。参数是可选的，也就是说，一个方法可能不包含参数。
- **Method body**：方法主体，包含了完成任务所需的指令集。

实例

下面的代码片段显示一个函数 *FindMax*，它接受两个整数值，并返回两个中的较大值。它有 public 访问修饰符，所以它可以使用类的实例从类的外部进行访问。

```
class NumberManipulator
{
   public int FindMax(int num1, int num2)
   {
      /* 局部变量声明 */
      int result;

      if (num1 > num2)
         result = num1;
      else
         result = num2;

      return result;
   }
   ...
}
```

C# 中调用方法

您可以使用方法名调用方法。下面的实例演示了这点：

```
using System;

namespace CalculatorApplication
{
   class NumberManipulator
   {
      public int FindMax(int num1, int num2)
      {
         /* 局部变量声明 */
         int result;

         if (num1 > num2)
            result = num1;
         else
            result = num2;

         return result;
      }
      static void Main(string[] args)
      {
         /* 局部变量定义 */
         int a = 100;
         int b = 200;
         int ret;
         NumberManipulator n = new NumberManipulator();

         //调用 FindMax 方法
         ret = n.FindMax(a, b);
         Console.WriteLine("最大值是： {0}", ret );
         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
最大值是： 200
```

您也可以使用类的实例从另一个类中调用其他类的公有方法。例如，方法 *FindMax* 属于 *NumberManipulator* 类，您可以从另一个类 *Test* 中调用它。

```
using System;

namespace CalculatorApplication
{
    class NumberManipulator
    {
        public int FindMax(int num1, int num2)
        {
            /* 局部变量声明 */
            int result;

            if (num1 > num2)
                result = num1;
            else
                result = num2;

            return result;
        }
    }
    class Test
    {
        static void Main(string[] args)
        {
            /* 局部变量定义 */
            int a = 100;
            int b = 200;
            int ret;
            NumberManipulator n = new NumberManipulator();
            //调用 FindMax 方法
            ret = n.FindMax(a, b);
            Console.WriteLine("最大值是： {0}", ret );
            Console.ReadLine();

        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
最大值是： 200
```

递归方法调用

一个方法可以自我调用。这就是所谓的 递归。下面的实例使用递归函数计算一个数的阶乘：

```
using System;

namespace CalculatorApplication
{
    class NumberManipulator
    {
        public int factorial(int num)
        {
            /* 局部变量定义 */
            int result;

            if (num == 1)
            {
                return 1;
            }
            else
            {
                result = factorial(num - 1) * num;
                return result;
            }
        }

        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
            //调用 factorial 方法
            Console.WriteLine("6 的阶乘是： {0}", n.factorial(6));
            Console.WriteLine("7 的阶乘是： {0}", n.factorial(7));
            Console.WriteLine("8 的阶乘是： {0}", n.factorial(8));
            Console.ReadLine();

        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
6 的阶乘是： 720
7 的阶乘是： 5040
8 的阶乘是： 40320
```

参数传递

当调用带有参数的方法时，您需要向方法传递参数。在 C# 中，有三种向方法传递参数的方式：

方式	描述
值参数	这种方式复制参数的实际值给函数的形式参数，实参和形参使用的是两个不同内存中的值。在这种情况下，当形参的值发生改变时，不会影响实参的值，从而保证了实参数据的安全。
引用参数	这种方式复制参数的内存位置的引用给形式参数。这意味着，当形参的值发生改变时，同时也改变实参的值。
输出参数	这种方式可以返回多个值。

按值传递参数

这是参数传递的默认方式。在这种方式下，当调用一个方法时，会为每个值参数创建一个新的存储位置。

实际参数的值会复制给形参，实参和形参使用的是两个不同内存中的值。所以，当形参的值发生改变时，不会影响实参的值，从而保证了实参数据的安全。下面的实例演示了这个概念：

```
using System;

namespace CalculatorApplication
{
    class NumberManipulator
    {
        public void swap(int x, int y)
        {
            int temp;

            temp = x; /* 保存 x 的值 */
            x = y;    /* 把 y 赋值给 x */
            y = temp; /* 把 temp 赋值给 y */
        }

        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
            /* 局部变量定义 */
            int a = 100;
            int b = 200;

            Console.WriteLine("在交换之前，a 的值： {0}", a);
            Console.WriteLine("在交换之前，b 的值： {0}", b);

            /* 调用函数来交换值 */
            n.swap(a, b);

            Console.WriteLine("在交换之后，a 的值： {0}", a);
            Console.WriteLine("在交换之后，b 的值： {0}", b);

            Console.ReadLine();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
在交换之前，a 的值：100
在交换之前，b 的值：200
在交换之后，a 的值：100
在交换之后，b 的值：200
```

结果表明，即使在函数内改变了值，值也没有发生任何的变化。

按引用传递参数

引用参数是一个对变量的内存位置的引用。当按引用传递参数时，与值参数不同的是，它不会为这些参数创建一个新的存储位置。引用参数表示与提供给方法的实际参数具有相同的内存位置。

在 C# 中，使用 **ref** 关键字声明引用参数。下面的实例演示了这点：

```
using System;
namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void swap(ref int x, ref int y)
      {
         int temp;

         temp = x; /* 保存 x 的值 */
         x = y;    /* 把 y 赋值给 x */
         y = temp; /* 把 temp 赋值给 y */
       }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* 局部变量定义 */
         int a = 100;
         int b = 200;

         Console.WriteLine("在交换之前，a 的值： {0}", a);
         Console.WriteLine("在交换之前，b 的值： {0}", b);

         /* 调用函数来交换值 */
         n.swap(ref a, ref b);

         Console.WriteLine("在交换之后，a 的值： {0}", a);
         Console.WriteLine("在交换之后，b 的值： {0}", b);

         Console.ReadLine();

      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
在交换之前，a 的值：100
在交换之前，b 的值：200
在交换之后，a 的值：200
在交换之后，b 的值：100
```

结果表明，*swap* 函数内的值改变了，且这个改变可以在 *Main* 函数中反映出来。

按输出传递参数

return 语句可用于只从函数中返回一个值。但是，可以使用 输出参数 来从函数中返回两个值。输出参数会把方法输出的数据赋给自己，其他方面与引用参数相似。

下面的实例演示了这点：

```
using System;

namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void getValue(out int x )
      {
         int temp = 5;
         x = temp;
      }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* 局部变量定义 */
         int a = 100;

         Console.WriteLine("在方法调用之前，a 的值： {0}", a);

         /* 调用函数来获取值 */
         n.getValue(out a);

         Console.WriteLine("在方法调用之后，a 的值： {0}", a);
         Console.ReadLine();

      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
在方法调用之前，a 的值： 100
在方法调用之后，a 的值： 5
```

提供给输出参数的变量不需要赋值。当需要从一个参数没有指定初始值的方法中返回值时，输出参数特别有用。请看下面的实例，来理解这一点：

```
using System;

namespace CalculatorApplication
{
   class NumberManipulator
   {
      public void getValues(out int x, out int y )
      {
          Console.WriteLine("请输入第一个值： ");
          x = Convert.ToInt32(Console.ReadLine());
          Console.WriteLine("请输入第二个值： ");
          y = Convert.ToInt32(Console.ReadLine());
      }

      static void Main(string[] args)
      {
         NumberManipulator n = new NumberManipulator();
         /* 局部变量定义 */
         int a , b;

         /* 调用函数来获取值 */
         n.getValues(out a, out b);

         Console.WriteLine("在方法调用之后，a 的值： {0}", a);
         Console.WriteLine("在方法调用之后，b 的值： {0}", b);
         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果（取决于用户输入）：

```
请输入第一个值：
7
请输入第二个值：
8
在方法调用之后，a 的值： 7
在方法调用之后，b 的值： 8
```

C# 可空类型（Nullable）

C# 可空类型（Nullable）

C# 提供了一个特殊的数据类型，**nullable** 类型（可空类型），可空类型可以表示其基础值类型正常范围内的值，再加上一个 null 值。

例如，Nullable< Int32 >，读作"可空的 Int32"，可以被赋值为 -2,147,483,648 到 2,147,483,647 之间的任意值，也可以被赋值为 null 值。类似的，Nullable< bool > 变量可以被赋值为 true 或 false 或 null。

在处理数据库和其他包含可能未赋值的元素的数据类型时，将 null 赋值给数值类型或布尔型的功能特别有用。例如，数据库中的布尔型字段可以存储值 true 或 false，或者，该字段也可以未定义。

声明一个 **nullable** 类型（可空类型）的语法如下：

```
< data_type> ? <variable_name> = null;
```

下面的实例演示了可空数据类型的用法：

```
using System;
namespace CalculatorApplication
{
   class NullablesAtShow
   {
      static void Main(string[] args)
      {
         int? num1 = null;
         int? num2 = 45;
         double? num3 = new double?();
         double? num4 = 3.14157;

         bool? boolval = new bool?();

         // 显示值

         Console.WriteLine("显示可空类型的值： {0}, {1}, {2}, {3}",
                            num1, num2, num3, num4);
         Console.WriteLine("一个可空的布尔值： {0}", boolval);
         Console.ReadLine();

      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
显示可空类型的值： , 45,  , 3.14157
一个可空的布尔值：
```

Null 合并运算符（ ?? ）

Null 合并运算符用于定义可空类型和引用类型的默认值。Null 合并运算符为类型转换定义了一个预设值，以防可空类型的值为 Null。Null 合并运算符把操作数类型隐式转换为另一个可空（或不可空）的值类型的操作数的类型。

如果第一个操作数的值为 null，则运算符返回第二个操作数的值，否则返回第一个操作数的值。下面的实例演示了这点：

```
using System;
namespace CalculatorApplication
{
   class NullablesAtShow
   {

      static void Main(string[] args)
      {

         double? num1 = null;
         double? num2 = 3.14157;
         double num3;
         num3 = num1 ?? 5.34;
         Console.WriteLine("num3 的值： {0}", num3);
         num3 = num2 ?? 5.34;
         Console.WriteLine("num3 的值： {0}", num3);
         Console.ReadLine();

      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

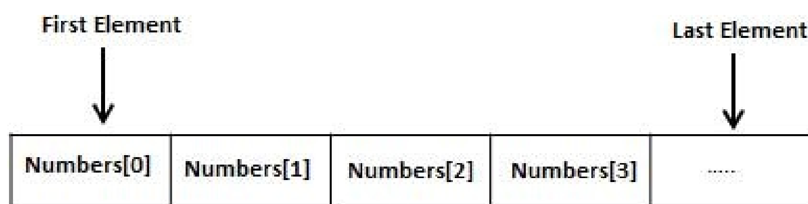
```
num3 的值： 5.34
num3 的值： 3.14157
```

C# 数组（Array）

数组是一个存储相同类型元素的固定大小的顺序集合。数组是用来存储数据的集合，通常认为数组是一个同一类型变量的集合。

声明数组变量并不是声明 number0、number1、...、number99 一个个单独的变量，而是声明一个就像 numbers 这样的变量，然后使用 numbers[0]、numbers[1]、...、numbers[99] 来表示一个个单独的变量。数组中某个指定的元素是通过索引来访问的。

所有的数组都是由连续的内存位置组成的。最低的地址对应第一个元素，最高的地址对应最后一个元素。



声明数组

在 C# 中声明一个数组，您可以使用下面的语法：

```
datatype[] arrayName;
```

其中，

- *datatype* 用于指定被存储在数组中的元素的类型。
- *[ ]* 指定数组的秩（维度）。秩指定数组的大小。
- *arrayName* 指定数组的名称。

例如：

```
double[] balance;
```

初始化数组

声明一个数组不会在内存中初始化数组。当初始化数组变量时，您可以赋值给数组。

数组是一个引用类型，所以您需要使用 **new** 关键字来创建数组的实例。

例如：

```
double[] balance = new double[10];
```

赋值给数组

您可以通过使用索引号赋值给一个单独的数组元素，比如：

```
double[] balance = new double[10];
balance[0] = 4500.0;
```

您可以在声明数组的同时给数组赋值，比如：

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

您也可以创建并初始化一个数组，比如：

```
int [] marks = new int[5]  { 99,  98, 92, 97, 95};
```

在上述情况下，你也可以省略数组的大小，比如：

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};
```

您也可以赋值一个数组变量到另一个目标数组变量中。在这种情况下，目标和源会指向相同的内存位置：

```
int [] marks = new int[]  { 99,  98, 92, 97, 95};
int[] score = marks;
```

当您创建一个数组时，C# 编译器会根据数组类型隐式初始化每个数组元素为一个默认值。例如，int 数组的所有元素都会被初始化为 0。

访问数组元素

元素是通过带索引的数组名称来访问的。这是通过把元素的索引放置在数组名称后的方括号中来实现的。例如：

```
double salary = balance[9];
```

下面是一个实例，使用上面提到的三个概念，即声明、赋值、访问数组：

```
using System;
namespace ArrayApplication
{
   class MyArray
   {
      static void Main(string[] args)
      {
         int []  n = new int[10]; /* n 是一个带有 10 个整数的数组 */
         int i,j;

         /* 初始化数组 n 中的元素 */
         for ( i = 0; i < 10; i++ )
         {
            n[ i ] = i + 100;
         }

         /* 输出每个数组元素的值 */
         for (j = 0; j < 10; j++ )
         {
            Console.WriteLine("Element[{0}] = {1}", j, n[j]);
         }
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

使用 *foreach* 循环

在前面的实例中，我们使用一个 for 循环来访问每个数组元素。您也可以使用一个 **foreach** 语句来遍历数组。

```
using System;

namespace ArrayApplication
{
   class MyArray
   {
      static void Main(string[] args)
      {
         int []  n = new int[10]; /* n 是一个带有 10 个整数的数组 */

         /* 初始化数组 n 中的元素 */
         for ( int i = 0; i < 10; i++ )
         {
            n[i] = i + 100;
         }

         /* 输出每个数组元素的值 */
         foreach (int j in n )
         {
            int i = j-100;
            Console.WriteLine("Element[{0}] = {1}", i, j);
            i++;
         }
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

C# 数组细节

在 C# 中，数组是非常重要的，且需要了解更多的细节。下面列出了 C# 程序员必须清楚的一些与数组相关的重要概念：

概念	描述
多维数组	C# 支持多维数组。多维数组最简单的形式是二维数组。
交错数组	C# 支持交错数组，即数组的数组。
传递数组给函数	您可以通过指定不带索引的数组名称来给函数传递一个指向数组的指针。
参数数组	这通常用于传递未知数量的参数给函数。
Array 类	在 System 命名空间中定义，是所有数组的基类，并提供了各种用于数组的属性和方法。

C# 字符串（String）

在 C# 中，您可以使用字符数组来表示字符串，但是，更常见的做法是使用 **string** 关键字来声明一个字符串变量。string 关键字是 **System.String** 类的别名。

创建 String 对象

您可以使用以下方法之一来穿件 string 对象：

- 通过给 String 变量指定一个字符串
- 通过使用 String 类构造函数
- 通过使用字符串串联运算符（ + ）
- 通过检索属性或调用一个返回字符串的方法
- 通过格式化方法来转换一个值或对象为它的字符串表示形式

下面的实例演示了这点：

```
using System;

namespace StringApplication
{
    class Program
    {
        static void Main(string[] args)
        {
           //字符串，字符串连接
            string fname, lname;
            fname = "Rowan";
            lname = "Atkinson";

            string fullname = fname + lname;
            Console.WriteLine("Full Name: {0}", fullname);

            //通过使用 string 构造函数
            char[] letters = { 'H', 'e', 'l', 'l','o' };
            string greetings = new string(letters);
            Console.WriteLine("Greetings: {0}", greetings);

            //方法返回字符串
            string[] sarray = { "Hello", "From", "Tutorials", "Point" };
            string message = String.Join(" ", sarray);
            Console.WriteLine("Message: {0}", message);

            //用于转化值的格式化方法
            DateTime waiting = new DateTime(2012, 10, 10, 17, 58, 1);
            string chat = String.Format("Message sent at {0:t} on {0:D}",
            waiting);
            Console.WriteLine("Message: {0}", chat);
            Console.ReadKey() ;
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Full Name: Rowan Atkinson
Greetings: Hello
Message: Hello From Tutorials Point
Message: Message sent at 5:58 PM on Wednesday, October 10, 2012
```

String 类的属性

String 类有以下两个属性：

名称	描述
Chars	在当前 *String* 对象中获取 *Char* 对象的指定位置。
Length	在当前的 *String* 对象中获取字符数。

String 类的方法

String 类有许多方法用于 string 对象的操作。下面的表格提供了一些最常用的方法：

名称	描述
public static int Compare(string strA, string strB)	比较两个指定的 string 对象，并返回一个表示它们在排列顺序中相对位置的整数。该方法区分大小写。
public static int Compare(string strA, string strB, bool ignoreCase)	比较两个指定的 string 对象，并返回一个表示它们在排列顺序中相对位置的整数。但是，如果布尔参数为真时，该方法不区分大小写。
public static string Concat(string str0, string str1)	连接两个 string 对象。
public static string Concat(string str0, string str1, string str2)	连接三个 string 对象。
public static string Concat(string str0, string str1, string str2, string str3)	连接四个 string 对象。
public bool Contains(string value)	返回一个表示指定 string 对象是否出现在字符串中的值。
public static string Copy(string str)	创建一个与指定字符串具有相同值的新的 String 对象。
public void CopyTo(int sourceIndex, char[] destination, int destinationIndex, int count)	从 string 对象的指定位置开始复制指定数量的字符到 Unicode 字符数组中的指定位置。
public bool EndsWith(	

string value)	
public bool Equals(string value)	判断当前的 string 对象是否与指定的 string 对象具有相同的值。
public static bool Equals(string a, string b)	判断两个指定的 string 对象是否具有相同的值。
public static string Format(string format, Object arg0)	把指定字符串中一个或多个格式项替换为指定对象的字符串表示形式。
public int IndexOf(char value)	返回指定 Unicode 字符在当前字符串中第一次出现的索引，索引从 0 开始。
public int IndexOf(string value)	返回指定字符串在该实例中第一次出现的索引，索引从 0 开始。
public int IndexOf(char value, int startIndex)	返回指定 Unicode 字符从该字符串中指定字符位置开始搜索第一次出现的索引，索引从 0 开始。
public int IndexOf(string value, int startIndex)	返回指定字符串从该实例中指定字符位置开始搜索第一次出现的索引，索引从 0 开始。
public int IndexOfAny(char[] anyOf)	返回某一个指定的 Unicode 字符数组中任意字符在该实例中第一次出现的索引，索引从 0 开始。
public int IndexOfAny(char[] anyOf, int startIndex)	返回某一个指定的 Unicode 字符数组中任意字符从该实例中指定字符位置开始搜索第一次出现的索引，索引从 0 开始。
public string Insert(int startIndex, string value)	返回一个新的字符串，其中，指定的字符串被插入在当前 string 对象的指定索引位置。
public static bool IsNullOrEmpty(string value)	指示指定的字符串是否为 null 或者是否为一个空的字符串。
public static string Join(string separator, params string[] value)	连接一个字符串数组中的所有元素，使用指定的分隔符分隔每个元素。
public static string Join(string separator, string[] value, int startIndex, int count)	链接一个字符串数组中的指定元素，使用指定的分隔符分隔每个元素。
public int LastIndexOf(char value)	返回指定 Unicode 字符在当前 string 对象中最后一次出现的索引位置，索引从 0 开始。
public int LastIndexOf(string value)	返回指定字符串在当前 string 对象中最后一次出现的索引位置，索引从 0 开始。
public string Remove(int startIndex)	移除当前实例中的所有字符，从指定位置开始，一直到最后一个位置为止，并返回字符串。
public string Remove(int startIndex, int count)	从当前字符串的指定位置开始移除指定数量的字符，并返回字符串。
public string Replace(char	把当前 string 对象中，所有指定的 Unicode 字符替换为

public string Replace(char oldChar, char newChar)	把当前 string 对象中，所有指定的 Unicode 字符替换为另一个指定的 Unicode 字符，并返回新的字符串。
public string Replace(string oldValue, string newValue)	把当前 string 对象中，所有指定的字符串替换为另一个指定的字符串，并返回新的字符串。
public string[] Split(params char[] separator)	返回一个字符串数组，包含当前的 string 对象中的子字符串，子字符串是使用指定的 Unicode 字符数组中的元素进行分隔的。
public string[] Split(char[] separator, int count)	返回一个字符串数组，包含当前的 string 对象中的子字符串，子字符串是使用指定的 Unicode 字符数组中的元素进行分隔的。int 参数指定要返回的子字符串的最大数目。
public bool StartsWith(string value)	判断字符串实例的开头是否匹配指定的字符串。
public char[] ToCharArray()	返回一个带有当前 string 对象中所有字符的 Unicode 字符数组。
public char[] ToCharArray(int startIndex, int length)	返回一个带有当前 string 对象中所有字符的 Unicode 字符数组，从指定的索引开始，直到指定的长度为止。
public string ToLower()	把字符串转换为小写并返回。
public string ToUpper()	把字符串转换为大写并返回。
public string Trim()	移除当前 String 对象中的所有前导空白字符和后置空白字符。

上面的方法列表并不详尽，请访问 MSDN 库，查看完整的方法列表和 String 类构造函数。

实例

下面的实例演示了上面提到的一些方法：

比较字符串

```
using System;

namespace StringApplication
{
   class StringProg
   {
      static void Main(string[] args)
      {
         string str1 = "This is test";
         string str2 = "This is text";

         if (String.Compare(str1, str2) == 0)
         {
            Console.WriteLine(str1 + " and " + str2 +  " are equal.");
         }
         else
         {
            Console.WriteLine(str1 + " and " + str2 + " are not equal.");
         }
         Console.ReadKey() ;
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
This is test and This is text are not equal.
```

字符串包含字符串：

```
using System;

namespace StringApplication
{
   class StringProg
   {
      static void Main(string[] args)
      {
         string str = "This is test";
         if (str.Contains("test"))
         {
            Console.WriteLine("The sequence 'test' was found.");
         }
         Console.ReadKey() ;
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
The sequence 'test' was found.
```

获取子字符串：

```
using System;

namespace StringApplication
{
   class StringProg
   {
      static void Main(string[] args)
      {
         string str = "Last night I dreamt of San Pedro";
         Console.WriteLine(str);
         string substr = str.Substring(23);
         Console.WriteLine(substr);
      }
      Console.ReadKey() ;
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
San Pedro
```

连接字符串：

```
using System;

namespace StringApplication
{
   class StringProg
   {
      static void Main(string[] args)
      {
         string[] starray = new string[]{"Down the way nights are dark",
         "And the sun shines daily on the mountain top",
         "I took a trip on a sailing ship",
         "And when I reached Jamaica",
         "I made a stop"};

         string str = String.Join("\n", starray);
         Console.WriteLine(str);
      }
      Console.ReadKey() ;
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Down the way nights are dark
And the sun shines daily on the mountain top
I took a trip on a sailing ship
And when I reached Jamaica
I made a stop
```

C# 结构（Struct）

在 C# 中，结构是值类型数据结构。它使得一个单一变量可以存储各种数据类型的相关数据。**struct** 关键字用于创建结构。

结构是用来代表一个记录。假设您想跟踪图书馆中书的动态。您可能想跟踪每本书的以下属性：

- Title
- Author
- Subject
- Book ID

定义结构

为了定义一个结构，您必须使用 struct 语句。struct 语句为程序定义了一个带有多个成员的新的数据类型。

例如，您可以按照如下的方式声明 Book 结构：

```
struct Books
{
   public string title;
   public string author;
   public string subject;
   public int book_id;
};
```

下面的程序演示了结构的用法：

```
using System;

struct Books
{
   public string title;
   public string author;
   public string subject;
   public int book_id;
};

public class testStructure
{
   public static void Main(string[] args)
   {

      Books Book1;        /* 声明 Book1，类型为 Book */
      Books Book2;        /* 声明 Book2，类型为 Book */

      /* book 1 详述 */
      Book1.title = "C Programming";
      Book1.author = "Nuha Ali";
      Book1.subject = "C Programming Tutorial";
      Book1.book_id = 6495407;

      /* book 2 详述 */
      Book2.title = "Telecom Billing";
      Book2.author = "Zara Ali";
      Book2.subject =  "Telecom Billing Tutorial";
      Book2.book_id = 6495700;

      /* 打印 Book1 信息 */
      Console.WriteLine( "Book 1 title : {0}", Book1.title);
      Console.WriteLine("Book 1 author : {0}", Book1.author);
      Console.WriteLine("Book 1 subject : {0}", Book1.subject);
      Console.WriteLine("Book 1 book_id :{0}", Book1.book_id);

      /* 打印 Book2 信息 */
      Console.WriteLine("Book 2 title : {0}", Book2.title);
      Console.WriteLine("Book 2 author : {0}", Book2.author);
      Console.WriteLine("Book 2 subject : {0}", Book2.subject);
      Console.WriteLine("Book 2 book_id : {0}", Book2.book_id);

      Console.ReadKey();

   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

C# 结构的特点

您已经用了一个简单的名为 Books 的结构。在 C# 中的结构与传统的 C 或 C++ 中的结构不同。C# 中的结构有一下特点：

- 结构可带有方法、字段、索引、属性、运算符方法和事件。
- 结构可定义构造函数，但不能定义析构函数。但是，您不能为结构定义默认的构造函数。默认的构造函数是自动定义的，且不能被改变。
- 与类不同，结构不能继承其他的结构或类。
- 结构不能作为其他结构或类的基础结构。
- 结构可实现一个或多个接口。
- 结构成员不能指定为 abstract、virtual 或 protected。
- 当您使用 **New** 操作符创建一个结构对象时，会调用适当的构造函数来创建结构。与类不同，结构可以不使用 New 操作符即可被实例化。
- 如果不使用 New 操作符，只有在所有的字段都被初始化之后，字段才被赋值，对象才被使用。

类 vs 结构

类和结构有以下几个基本的不同点：

- 类是引用类型，结构是值类型。
- 结构不支持继承。
- 结构不能声明默认的构造函数。

针对上述讨论，让我们重写前面的实例：

```
using System;

struct Books
{
   private string title;
   private string author;
   private string subject;
   private int book_id;
   public void getValues(string t, string a, string s, int id)
   {
      title = t;
      author = a;
      subject = s;
      book_id = id;
   }
   public void display()
   {
      Console.WriteLine("Title : {0}", title);
      Console.WriteLine("Author : {0}", author);
      Console.WriteLine("Subject : {0}", subject);
      Console.WriteLine("Book_id :{0}", book_id);
   }

};

public class testStructure
{
   public static void Main(string[] args)
   {

      Books Book1 = new Books(); /* 声明 Book1，类型为 Book */
      Books Book2 = new Books(); /* 声明 Book2，类型为 Book */

      /* book 1 详述 */
      Book1.getValues("C Programming",
      "Nuha Ali", "C Programming Tutorial",6495407);

      /* book 2 详述 */
      Book2.getValues("Telecom Billing",
      "Zara Ali", "Telecom Billing Tutorial", 6495700);

      /* 打印 Book1 信息 */
      Book1.display();

      /* 打印 Book2 信息 */
      Book2.display();

      Console.ReadKey();

   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Title : C Programming
Author : Nuha Ali
Subject : C Programming Tutorial
Book_id : 6495407
Title : Telecom Billing
Author : Zara Ali
Subject : Telecom Billing Tutorial
Book_id : 6495700
```

C# 枚举（Enum）

枚举是一组命名整型常量。枚举类型是使用 **enum** 关键字声明的。

C# 枚举是值数据类型。换句话说，枚举包含自己的值，且不能继承或传递继承。

声明 *enum* 变量

声明枚举的一般语法：

```
enum <enum_name>
{
    enumeration list
};
```

其中，

- *enum_name* 指定枚举的类型名称。
- *enumeration list* 是一个用逗号分隔的标识符列表。

枚举列表中的每个符号代表一个整数值，一个比它前面的符号大的整数值。默认情况下，第一个枚举符号的值是 0.例如：

```
enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
```

实例

下面的实例演示了枚举变量的用法：

```
using System;
namespace EnumApplication
{
   class EnumProgram
   {
      enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };

      static void Main(string[] args)
      {
         int WeekdayStart = (int)Days.Mon;
         int WeekdayEnd = (int)Days.Fri;
         Console.WriteLine("Monday: {0}", WeekdayStart);
         Console.WriteLine("Friday: {0}", WeekdayEnd);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Monday: 1
Friday: 5
```

C# 类（Class）

当您定义一个类时，您定义了一个数据类型的蓝图。这实际上并没有定义任何的数据，但它定义了类的名称意味着什么，也就是说，类的对象由什么组成及在这个对象上可执行什么操作。对象是类的实例。构成类的方法和变量成为类的成员。

类的定义

类的定义是以关键字 **class** 开始，后跟类的名称。类的主体，包含在一对花括号内。下面是类定义的一般形式：

```
<access specifier> class  class_name
{
    // member variables
    <access specifier> <data type> variable1;
    <access specifier> <data type> variable2;
    ...
    <access specifier> <data type> variableN;
    // member methods
    <access specifier> <return type> method1(parameter_list)
    {
        // method body
    }
    <access specifier> <return type> method2(parameter_list)
    {
        // method body
    }
    ...
    <access specifier> <return type> methodN(parameter_list)
    {
        // method body
    }
}
```

请注意：

- 访问标识符 <access specifier> 指定了对类及其成员的访问规则。如果没有指定，则使用默认的访问标识符。类的默认访问标识符是 **internal**，成员的默认访问标识符是 **private**。
- 数据类型 <data type> 指定了变量的类型，返回类型 <return type> 指定了返回的方法返回的数据类型。
- 如果要访问类的成员，您要使用点（.）运算符。
- 点运算符链接了对象的名称和成员的名称。

下面的实例说明了目前为止所讨论的概念：

```
using System;
namespace BoxApplication
{
    class Box
    {
       public double length;   // 长度
       public double breadth;  // 宽度
       public double height;   // 高度
    }
    class Boxtester
    {
        static void Main(string[] args)
        {
            Box Box1 = new Box();        // 声明 Box1，类型为 Box
            Box Box2 = new Box();        // 声明 Box2，类型为 Box
            double volume = 0.0;         // 体积

            // Box1 详述
            Box1.height = 5.0;
            Box1.length = 6.0;
            Box1.breadth = 7.0;

            // Box2 详述
            Box2.height = 10.0;
            Box2.length = 12.0;
            Box2.breadth = 13.0;

            // Box1 的体积
            volume = Box1.height * Box1.length * Box1.breadth;
            Console.WriteLine("Box1 的体积： {0}",  volume);

            // Box2 的体积
            volume = Box2.height * Box2.length * Box2.breadth;
            Console.WriteLine("Box2 的体积： {0}", volume);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1 的体积： 210
Box2 的体积： 1560
```

成员函数和封装

类的成员函数是一个在类定义中有它的定义或原型的函数，就像其他变量一样。作为类的一个成为，它能在类的任何对象上操作，且能访问该对象的类的所有成员。

成员变量是对象的属性（从设计角度），且它们保持私有来实现封装。这些变量只能使用公共成员函数来访问。

让我们使用上面的概念来设置和获取一个类中不同的类成员的值：

```
using System;
namespace BoxApplication
{
    class Box
    {
       private double length;   // 长度
       private double breadth;  // 宽度
       private double height;   // 高度
       public void setLength( double len )
       {
            length = len;
       }

       public void setBreadth( double bre )
       {
            breadth = bre;
       }

       public void setHeight( double hei )
       {
            height = hei;
       }
       public double getVolume()
       {
           return length * breadth * height;
       }
    }
    class Boxtester
    {
        static void Main(string[] args)
        {
            Box Box1 = new Box();        // 声明 Box1，类型为 Box
            Box Box2 = new Box();         // 声明 Box2，类型为 Box
            double volume;                 // 体积

            // Box1 详述
            Box1.setLength(6.0);
            Box1.setBreadth(7.0);
            Box1.setHeight(5.0);

            // Box2 详述
            Box2.setLength(12.0);
            Box2.setBreadth(13.0);
            Box2.setHeight(10.0);

            // Box1 的体积
            volume = Box1.getVolume();
            Console.WriteLine("Box1 的体积： {0}" ,volume);

            // Box2 的体积
            volume = Box2.getVolume();
            Console.WriteLine("Box2 的体积： {0}", volume);

            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1 的体积： 210
Box2 的体积： 1560
```

C# 中的构造函数

类的 构造函数 是类的一个特殊的成员函数，当创建类的新对象时执行。

构造函数的名称与类的名称完全相同，它没有任何返回类型。

下面的实例说明了构造函数的概念：

```
using System;
namespace LineApplication
{
   class Line
   {
      private double length;   // 线条的长度
      public Line()
      {
         Console.WriteLine("对象已创建");
      }

      public void setLength( double len )
      {
         length = len;
      }
      public double getLength()
      {
         return length;
      }

      static void Main(string[] args)
      {
         Line line = new Line();
         // 设置线条长度
         line.setLength(6.0);
         Console.WriteLine("线条的长度： {0}", line.getLength());
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
对象已创建
线条的长度： 6
```

默认的构造函数没有任何参数。但是如果您需要一个带有参数的构造函数可以有参数，这种构造函数叫做参数化构造函数。这种技术可以帮助您在创建对象的同时给对象赋初始值，具体请看下面实例：

```
using System;
namespace LineApplication
{
   class Line
   {
      private double length;   // 线条的长度
      public Line(double len)  // 参数化构造函数
      {
         Console.WriteLine("对象已创建，length = {0}", len);
         length = len;
      }

      public void setLength( double len )
      {
         length = len;
      }
      public double getLength()
      {
         return length;
      }

      static void Main(string[] args)
      {
         Line line = new Line(10.0);
         Console.WriteLine("线条的长度： {0}", line.getLength());
         // 设置线条长度
         line.setLength(6.0);
         Console.WriteLine("线条的长度： {0}", line.getLength());
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
对象已创建，length = 10
线条的长度： 10
线条的长度： 6
```

C# 中的析构函数

类的 析构函数 是类的一个特殊的成员函数，当类的对象超出范围时执行。

析构函数的名称是在类的名称前加上一个波浪形（~）作为前缀，它不返回值，也不带任何参数。

析构函数用于在结束程序（比如关闭文件、释放内存等）之前释放资源。析构函数不能继承或重载。

下面的实例说明了析构函数的概念：

```
using System;
namespace LineApplication
{
   class Line
   {
      private double length;   // 线条的长度
      public Line()  // 构造函数
      {
         Console.WriteLine("对象已创建");
      }
      ~Line() //析构函数
      {
         Console.WriteLine("对象已删除");
      }

      public void setLength( double len )
      {
         length = len;
      }
      public double getLength()
      {
         return length;
      }

      static void Main(string[] args)
      {
         Line line = new Line();
         // 设置线条长度
         line.setLength(6.0);
         Console.WriteLine("线条的长度： {0}", line.getLength());
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
对象已创建
线条的长度： 6
对象已删除
```

C# 类的静态成员

我们可以使用 **static** 关键字把类成员定义为静态的。当我们声明一个类成员为静态时，意味着无论有多少个类的对象被创建，只会有一个该静态成员的副本。

关键字 **static** 意味着类中只有一个该成员的实例。静态变量用于定义常量，因为它们的值可以通过直接调用类而不需要创建类的实例来获取。静态变量可在成员函数或类的定义外部进行初始化。您也可以在类的定义内部初始化静态变量。

下面的实例演示了静态变量的用法：

```
using System;
namespace StaticVarApplication
{
    class StaticVar
    {
       public static int num;
        public void count()
        {
            num++;
        }
        public int getNum()
        {
            return num;
        }
    }
    class StaticTester
    {
        static void Main(string[] args)
        {
            StaticVar s1 = new StaticVar();
            StaticVar s2 = new StaticVar();
            s1.count();
            s1.count();
            s1.count();
            s2.count();
            s2.count();
            s2.count();
            Console.WriteLine("s1 的变量 num： {0}", s1.getNum());
            Console.WriteLine("s2 的变量 num： {0}", s2.getNum());
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
s1 的变量 num： 6
s2 的变量 num： 6
```

您也可以把一个成员函数声明为 **static**。这样的函数只能访问静态变量。静态函数在对象被创建之前就已经存在。下面的实例演示了静态函数的用法：

```
using System;
namespace StaticVarApplication
{
    class StaticVar
    {
       public static int num;
        public void count()
        {
            num++;
        }
        public static int getNum()
        {
            return num;
        }
    }
    class StaticTester
    {
        static void Main(string[] args)
        {
            StaticVar s = new StaticVar();
            s.count();
            s.count();
            s.count();
            Console.WriteLine("变量 num： {0}", StaticVar.getNum());
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
变量 num： 3
```

C# 继承

继承是面向对象程序设计中最重要的概念之一。继承允许我们根据一个类来定义另一个类来定义一个类，这使得创建和维护应用程序变得更容易。同时也有利于重用代码和节省开发时间。

当创建一个类时，程序员不需要完全重新编写新的数据成员和成员函数，只需要设计一个新的类，继承了已有的类的成员即可。这个已有的类被称为的基类，这个新的类被称为派生类。

继承的思想实现了 属于（**IS-A**） 关系。例如，哺乳动物 属于（**IS-A**） 动物，狗 属于（**IS-A**） 哺乳动物，因此狗 属于（**IS-A**） 动物。

基类和派生类

一个类可以派生自多个类或接口，这意味着它可以从多个基类或接口继承数据和函数。

C# 中创建派生类的语法如下：

```
<acess-specifier> class <base_class>
{
 ...
}
class <derived_class> : <base_class>
{
 ...
}
```

假设，有一个基类 Shape，它的派生类是 Rectangle：

```
using System;
namespace InheritanceApplication
{
   class Shape
   {
      public void setWidth(int w)
      {
         width = w;
      }
      public void setHeight(int h)
      {
         height = h;
      }
      protected int width;
      protected int height;
   }

   // 派生类
   class Rectangle: Shape
   {
      public int getArea()
      {
         return (width * height);
      }
   }

   class RectangleTester
   {
      static void Main(string[] args)
      {
         Rectangle Rect = new Rectangle();

         Rect.setWidth(5);
         Rect.setHeight(7);

         // 打印对象的面积
         Console.WriteLine("总面积： {0}",  Rect.getArea());
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
总面积： 35
```

基类的初始化

派生类继承了基类的成员变量和成员方法。因此父类对象应在子类对象创建之前被创建。您可以在成员初始化列表中进行父类的初始化。

下面的程序演示了这点：

```
using System;
namespace RectangleApplication
{
   class Rectangle
   {
      // 成员变量
      protected double length;
      protected double width;
      public Rectangle(double l, double w)
      {
         length = l;
         width = w;
      }
      public double GetArea()
      {
         return length * width;
      }
      public void Display()
      {
         Console.WriteLine("长度： {0}", length);
         Console.WriteLine("宽度： {0}", width);
         Console.WriteLine("面积： {0}", GetArea());
      }
   }//end class Rectangle
   class Tabletop : Rectangle
   {
      private double cost;
      public Tabletop(double l, double w) : base(l, w)
      { }
      public double GetCost()
      {
         double cost;
         cost = GetArea() * 70;
         return cost;
      }
      public void Display()
      {
         base.Display();
         Console.WriteLine("成本： {0}", GetCost());
      }
   }
   class ExecuteRectangle
   {
      static void Main(string[] args)
      {
         Tabletop t = new Tabletop(4.5, 7.5);
         t.Display();
         Console.ReadLine();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
长度： 4.5
宽度： 7.5
面积： 33.75
成本： 2362.5
```

C# 多重继承

C# 不支持多重继承。但是，您可以使用接口来实现多重继承。下面的程序演示了这点：

```
using System;
namespace InheritanceApplication
{
   class Shape
   {
      public void setWidth(int w)
      {
         width = w;
      }
      public void setHeight(int h)
      {
         height = h;
      }
      protected int width;
      protected int height;
   }

   // 基类 PaintCost
   public interface PaintCost
   {
      int getCost(int area);

   }
   // 派生类
   class Rectangle : Shape, PaintCost
   {
      public int getArea()
      {
         return (width * height);
      }
      public int getCost(int area)
      {
         return area * 70;
      }
   }
   class RectangleTester
   {
      static void Main(string[] args)
      {
         Rectangle Rect = new Rectangle();
         int area;
         Rect.setWidth(5);
         Rect.setHeight(7);
         area = Rect.getArea();
         // 打印对象的面积
         Console.WriteLine("总面积： {0}",  Rect.getArea());
         Console.WriteLine("油漆总成本： ${0}" , Rect.getCost(area));
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
总面积： 35
油漆总成本： $2450
```

C# 多态性

多态性意味着有多重形式。在面向对象编程范式中，多态性往往表现为"一个接口，多个功能"。

多态性可以是静态的或动态的。在静态多态性中，函数的响应是在编译时发生的。在动态多态性中，函数的响应是在运行时发生的。

静态多态性

在编译时，函数和对象的连接机制被称为早期绑定，也被称为静态绑定。C# 提供了两种技术来实现静态多态性。分别为：

- 函数重载
- 运算符重载

运算符重载将在下一章节讨论，接下来我们将讨论函数重载。

函数重载

您可以在同一个范围内对相同的函数名有多个定义。函数的定义必须彼此不同，可以是参数列表中的参数类型不同，也可以是参数个数不同。不能重载只有返回类型不同的函数声明。

下面的实例演示了几个相同的函数 **print()**，用于打印不同的数据类型：

```
using System;
namespace PolymorphismApplication
{
   class Printdata
   {
      void print(int i)
      {
         Console.WriteLine("Printing int: {0}", i );
      }

      void print(double f)
      {
         Console.WriteLine("Printing float: {0}" , f);
      }

      void print(string s)
      {
         Console.WriteLine("Printing string: {0}", s);
      }
      static void Main(string[] args)
      {
         Printdata p = new Printdata();
         // 调用 print 来打印整数
         p.print(5);
         // 调用 print 来打印浮点数
         p.print(500.263);
         // 调用 print 来打印字符串
         p.print("Hello C++");
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Printing int: 5
Printing float: 500.263
Printing string: Hello C++
```

动态多态性

C# 允许您使用关键字 **abstract** 创建抽象类，用于提供接口的部分类的实现。当一个派生类继承自该抽象类时，实现即完成。抽象类包含抽象方法，抽象方法可被派生类实现。派生类具有更专业的功能。

请注意，下面是有关抽象类的一些规则：

- 您不能创建一个抽象类的实例。
- 您不能在一个抽象类外部声明一个抽象方法。
- 通过在类定义前面放置关键字 **sealed**，可以将类声明为密封类。当一个类被声明为 **sealed** 时，它不能被继承。抽象类不能被声明为 sealed。

下面的程序演示了一个抽象类：

```
using System;
namespace PolymorphismApplication
{
   abstract class Shape
   {
      public abstract int area();
   }
   class Rectangle:  Shape
   {
      private int length;
      private int width;
      public Rectangle( int a=0, int b=0)
      {
         length = a;
         width = b;
      }
      public override int area ()
      {
         Console.WriteLine("Rectangle 类的面积：");
         return (width * length);
      }
   }

   class RectangleTester
   {
      static void Main(string[] args)
      {
         Rectangle r = new Rectangle(10, 7);
         double a = r.area();
         Console.WriteLine("面积： {0}",a);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Rectangle 类的面积：
面积： 70
```

当有一个定义在类中的函数需要在继承类中实现时，可以使用虚方法。虚方法是使用关键字 **virtual** 声明的。虚方法可以在不同的继承类中有不同的实现。对虚方法的调用是在运行时发生的。

动态多态性是通过 抽象类 和 虚方法 实现的。

下面的程序演示了这点：

```
using System;
namespace PolymorphismApplication
{
   class Shape
   {
      protected int width, height;
      public Shape( int a=0, int b=0)
      {
         width = a;
         height = b;
      }
      public virtual int area()
      {
         Console.WriteLine("父类的面积：");
         return 0;
      }
   }
   class Rectangle: Shape
   {
      public Rectangle( int a=0, int b=0): base(a, b)
      {

      }
      public override int area ()
      {
         Console.WriteLine("Rectangle 类的面积：");
         return (width * height);
      }
   }
   class Triangle: Shape
   {
      public Triangle(int a = 0, int b = 0): base(a, b)
      {

      }
      public override int area()
      {
         Console.WriteLine("Triangle 类的面积：");
         return (width * height / 2);
      }
   }
   class Caller
   {
      public void CallArea(Shape sh)
      {
         int a;
         a = sh.area();
         Console.WriteLine("面积： {0}", a);
      }
   }
   class Tester
   {

      static void Main(string[] args)
      {
         Caller c = new Caller();
         Rectangle r = new Rectangle(10, 7);
         Triangle t = new Triangle(10, 5);
         c.CallArea(r);
         c.CallArea(t);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Rectangle 类的面积：
面积：70
Triangle 类的面积：
面积：25
```

C# 运算符重载

您可以重定义或重载 C# 中内置的运算符。因此，程序员也可以使用用户自定义类型的运算符。重载运算符是具有特殊名称的函数，是通过关键字 **operator** 后跟运算符的符号来定义的。与其他函数一样，重载运算符有返回类型和参数列表。

例如，请看下面的函数：

```
public static Box operator+ (Box b, Box c)
{
   Box box = new Box();
   box.length = b.length + c.length;
   box.breadth = b.breadth + c.breadth;
   box.height = b.height + c.height;
   return box;
}
```

上面的函数为用户自定义的类 Box 实现了加法运算符（+）。它把两个 Box 对象的属性相加，并返回相加后的 Box 对象。

运算符重载的实现

下面的程序演示了完整的实现：

```
using System;

namespace OperatorOvlApplication
{
   class Box
   {
      private double length;      // 长度
      private double breadth;     // 宽度
      private double height;      // 高度

      public double getVolume()
      {
         return length * breadth * height;
      }
      public void setLength( double len )
      {
         length = len;
      }

      public void setBreadth( double bre )
      {
         breadth = bre;
      }

      public void setHeight( double hei )
      {
         height = hei;
      }
      // 重载 + 运算符来把两个 Box 对象相加
      public static Box operator+ (Box b, Box c)
      {
         Box box = new Box();
```

```
         box.length = b.length + c.length;
         box.breadth = b.breadth + c.breadth;
         box.height = b.height + c.height;
         return box;
      }

   }

   class Tester
   {
      static void Main(string[] args)
      {
         Box Box1 = new Box();         // 声明 Box1，类型为 Box
         Box Box2 = new Box();         // 声明 Box2，类型为 Box
         Box Box3 = new Box();         // 声明 Box3，类型为 Box
         double volume = 0.0;          // 体积

         // Box1 详述
         Box1.setLength(6.0);
         Box1.setBreadth(7.0);
         Box1.setHeight(5.0);

         // Box2 详述
         Box2.setLength(12.0);
         Box2.setBreadth(13.0);
         Box2.setHeight(10.0);

         // Box1 的体积
         volume = Box1.getVolume();
         Console.WriteLine("Box1 的体积： {0}", volume);

         // Box2 的体积
         volume = Box2.getVolume();
         Console.WriteLine("Box2 的体积： {0}", volume);

         // 把两个对象相加
         Box3 = Box1 + Box2;

         // Box3 的体积
         volume = Box3.getVolume();
         Console.WriteLine("Box3 的体积： {0}", volume);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1 的体积： 210
Box2 的体积： 1560
Box3 的体积： 5400
```

可重载和不可重载运算符

下表描述了 C# 中运算符重载的能力：

运算符	描述
+, -, !, ~, ++, --	这些一元运算符只有一个操作数，且可以被重载。
+, -, *, /, %	这些二元运算符带有两个操作数，且可以被重载。
==, !=, <, >, <=, >=	这些比较运算符可以被重载。
&&, ||	这些条件逻辑运算符不能被直接重载。
+=, -=, *=, /=, %=	这些赋值运算符不能被重载。
=, ., ?:, ->, new, is, sizeof, typeof	这些运算符不能被重载。

实例

针对上述讨论，让我们扩展上面的实例，重载更多的运算符：

```
using System;

namespace OperatorOvlApplication
{
    class Box
    {
       private double length;      // 长度
       private double breadth;     // 宽度
       private double height;      // 高度

       public double getVolume()
       {
         return length * breadth * height;
       }
      public void setLength( double len )
      {
          length = len;
      }

      public void setBreadth( double bre )
      {
          breadth = bre;
      }

      public void setHeight( double hei )
      {
          height = hei;
      }
      // 重载 + 运算符来把两个 Box 对象相加
      public static Box operator+ (Box b, Box c)
      {
          Box box = new Box();
          box.length = b.length + c.length;
          box.breadth = b.breadth + c.breadth;
          box.height = b.height + c.height;
          return box;
      }

      public static bool operator == (Box lhs, Box rhs)
      {
          bool status = false;
          if (lhs.length == rhs.length && lhs.height == rhs.height
             && lhs.breadth == rhs.breadth)
          {
              status = true;
          }
```

```
            return status;
        }
        public static bool operator !=(Box lhs, Box rhs)
        {
            bool status = false;
            if (lhs.length != rhs.length || lhs.height != rhs.height
                || lhs.breadth != rhs.breadth)
            {
                status = true;
            }
            return status;
        }
        public static bool operator <(Box lhs, Box rhs)
        {
            bool status = false;
            if (lhs.length < rhs.length && lhs.height
                < rhs.height && lhs.breadth < rhs.breadth)
            {
                status = true;
            }
            return status;
        }

        public static bool operator >(Box lhs, Box rhs)
        {
            bool status = false;
            if (lhs.length > rhs.length && lhs.height
                > rhs.height && lhs.breadth > rhs.breadth)
            {
                status = true;
            }
            return status;
        }

        public static bool operator <=(Box lhs, Box rhs)
        {
            bool status = false;
            if (lhs.length <= rhs.length && lhs.height
                <= rhs.height && lhs.breadth <= rhs.breadth)
            {
                status = true;
            }
            return status;
        }

        public static bool operator >=(Box lhs, Box rhs)
        {
            bool status = false;
            if (lhs.length >= rhs.length && lhs.height
               >= rhs.height && lhs.breadth >= rhs.breadth)
            {
                status = true;
            }
            return status;
        }
        public override string ToString()
        {
            return String.Format("({0}, {1}, {2})", length, breadth, height);
        }

    }

    class Tester
    {
        static void Main(string[] args)
        {
          Box Box1 = new Box();          // 声明 Box1，类型为 Box
          Box Box2 = new Box();          // 声明 Box2，类型为 Box
          Box Box3 = new Box();          // 声明 Box3，类型为 Box
          Box Box4 = new Box();
          double volume = 0.0;   // 体积
```

```
         // Box1 详述
         Box1.setLength(6.0);
         Box1.setBreadth(7.0);
         Box1.setHeight(5.0);

         // Box2 详述
         Box2.setLength(12.0);
         Box2.setBreadth(13.0);
         Box2.setHeight(10.0);

        // 使用重载的 ToString() 显示两个盒子
         Console.WriteLine("Box1： {0}", Box1.ToString());
         Console.WriteLine("Box2： {0}", Box2.ToString());

         // Box1 的体积
         volume = Box1.getVolume();
         Console.WriteLine("Box1 的体积： {0}", volume);

         // Box2 的体积
         volume = Box2.getVolume();
         Console.WriteLine("Box2 的体积： {0}", volume);

         // 把两个对象相加
         Box3 = Box1 + Box2;
         Console.WriteLine("Box3： {0}", Box3.ToString());
         // Box3 的体积
         volume = Box3.getVolume();
         Console.WriteLine("Box3 的体积： {0}", volume);

         //comparing the boxes
         if (Box1 > Box2)
           Console.WriteLine("Box1 大于 Box2");
         else
           Console.WriteLine("Box1 不大于 Box2");
         if (Box1 < Box2)
           Console.WriteLine("Box1 小于 Box2");
         else
           Console.WriteLine("Box1 不小于 Box2");
         if (Box1 >= Box2)
           Console.WriteLine("Box1 大于等于 Box2");
         else
           Console.WriteLine("Box1 不大于等于 Box2");
         if (Box1 <= Box2)
           Console.WriteLine("Box1 小于等于 Box2");
         else
           Console.WriteLine("Box1 不小于等于 Box2");
         if (Box1 != Box2)
           Console.WriteLine("Box1 不等于 Box2");
         else
           Console.WriteLine("Box1 等于 Box2");
         Box4 = Box3;
         if (Box3 == Box4)
           Console.WriteLine("Box3 等于 Box4");
         else
           Console.WriteLine("Box3 不等于 Box4");

         Console.ReadKey();
      }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Box1： (6, 7, 5)
Box2： (12, 13, 10)
Box1 的体积： 210
Box2 的体积： 1560
Box3： (18, 20, 15)
Box3 的体积： 5400
Box1 不大于 Box2
Box1 小于 Box2
Box1 不大于等于 Box2
Box1 小于等于 Box2
Box1 不等于 Box2
Box3 等于 Box4
```

C# 接口（Interface）

接口定义了所有类继承接口时应遵循的语法合同。接口定义了语法合同 **"是什么"** 部分，派生类定义了语法合同 **"怎么做"** 部分。

接口定义了属性、方法和事件，这些都是接口的成员。接口只包含了成员的声明。成员的定义是派生类的责任。接口提供了派生类应遵循的标准结构。

抽象类在某种程度上与接口类似，但是，它们大多只是用在当只有少数方法由基类声明由派生类实现时。

声明接口

接口使用 **interface** 关键字声明，它与类的声明类似。接口声明默认是 public 的。下面是一个接口声明的实例：

```
public interface ITransactions
{
   // 接口成员
   void showTransaction();
   double getAmount();
}
```

实例

下面的实例演示了上面接口的实现：

```
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace InterfaceApplication
{

   public interface ITransactions
   {
      // 接口成员
      void showTransaction();
      double getAmount();
   }
   public class Transaction : ITransactions
   {
      private string tCode;
      private string date;
      private double amount;
      public Transaction()
      {
         tCode = " ";
         date = " ";
         amount = 0.0;
      }
      public Transaction(string c, string d, double a)
      {
         tCode = c;
         date = d;
         amount = a;
      }
      public double getAmount()
      {
         return amount;
      }
      public void showTransaction()
      {
         Console.WriteLine("Transaction: {0}", tCode);
         Console.WriteLine("Date: {0}", date);
         Console.WriteLine("Amount: {0}", getAmount());

      }

   }
   class Tester
   {
      static void Main(string[] args)
      {
         Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);
         Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);
         t1.showTransaction();
         t2.showTransaction();
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Transaction: 001
Date: 8/10/2012
Amount: 78900
Transaction: 002
Date: 9/10/2012
Amount: 451900
```

C# 命名空间（Namespace）

命名空间的设计目的是为了提供一种让一组名称与其他名称分隔开的方式。在一个命名空间中声明的类的名称与另一个命名空间中声明的相同的类的名称不冲突。

定义命名空间

命名空间的定义是以关键字 **namespace** 开始，后跟命名空间的名称，如下所示：

```
namespace namespace_name
{
   // 代码声明
}
```

为了调用支持命名空间版本的函数或变量，会把命名空间的名称置于前面，如下所示：

```
namespace_name.item_name;
```

下面的程序演示了命名空间的用法：

```
using System;
namespace first_space
{
   class namespace_cl
   {
      public void func()
      {
         Console.WriteLine("Inside first_space");
      }
   }
}
namespace second_space
{
   class namespace_cl
   {
      public void func()
      {
         Console.WriteLine("Inside second_space");
      }
   }
}
class TestClass
{
   static void Main(string[] args)
   {
      first_space.namespace_cl fc = new first_space.namespace_cl();
      second_space.namespace_cl sc = new second_space.namespace_cl();
      fc.func();
      sc.func();
      Console.ReadKey();
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Inside first_space
Inside second_space
```

using 关键字

using 关键字表明程序使用的是给定命名空间中的名称。例如，我们在程序中使用 **System** 命名空间，其中定义了类 Console。我们可以只写：

```
Console.WriteLine ("Hello there");
```

我们可以写完全限定名称，如下：

```
System.Console.WriteLine("Hello there");
```

您也可以使用 **using** 命名空间指令，这样在使用的时候就不用在前面加上命名空间名称。该指令告诉编译器随后的代码使用了指定命名空间中的名称。下面的代码延时了命名空间的应用。

让我们使用 using 指定重写上面的实例：

```
using System;
using first_space;
using second_space;

namespace first_space
{
   class abc
   {
      public void func()
      {
         Console.WriteLine("Inside first_space");
      }
   }
}
namespace second_space
{
   class efg
   {
      public void func()
      {
         Console.WriteLine("Inside second_space");
      }
   }
}
class TestClass
{
   static void Main(string[] args)
   {
      abc fc = new abc();
      efg sc = new efg();
      fc.func();
      sc.func();
      Console.ReadKey();
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Inside first_space
Inside second_space
```

嵌套命名空间

命名空间可以被嵌套，即您可以在一个命名空间内定义另一个命名空间，如下所示：

```
namespace namespace_name1
{
   // 代码声明
   namespace namespace_name2
   {
     // 代码声明
   }
}
```

您可以使用点（.）运算符访问嵌套的命名空间的成员，如下所示：

```
using System;
using first_space;
using first_space.second_space;

namespace first_space
{
   class abc
   {
      public void func()
      {
         Console.WriteLine("Inside first_space");
      }
   }
   namespace second_space
   {
      class efg
      {
         public void func()
         {
            Console.WriteLine("Inside second_space");
         }
      }
   }
}

class TestClass
{
   static void Main(string[] args)
   {
      abc fc = new abc();
      efg sc = new efg();
      fc.func();
      sc.func();
      Console.ReadKey();
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Inside first_space
Inside second_space
```

C# 预处理器指令

预处理器指令指导编译器在实际编译开始之前对信息进行预处理。

所有的预处理器指令都是以 # 开始。且在一行上，只有空白字符可以出现在预处理器指令之前。预处理器指令不是语句，所以它们不以分号（;）结束。

C# 编译器没有一个单独的预处理器，但是，指令被处理时就像是有一个单独的预处理器一样。在 C# 中，预处理器指令用于在条件编译中起作用。与 C 和 C++ 不同指令不用，它们不是用来创建宏。一个预处理器指令必须是该行上的唯一指令。

C# 预处理器指令列表

下表列出了 C# 中可用的预处理器指令：

预处理器指令	描述
#define	它用于定义一系列成为符号的字符。
#undef	它用于取消定义符号。
#if	它用于测试符号是否为真。
#else	它用于创建复合条件指令，与 #if 一起使用。
#elif	它用于创建复合条件指令。
#endif	指定一个条件指令的结束。
#line	它可以让您修改编译器的行数以及（可选地）输出错误和警告的文件名。
#error	它允许从代码的指定位置生成一个错误。
#warning	它允许从代码的指定位置生成一级警告。
#region	它可以让您在使用 Visual Studio Code Editor 的大纲特性时，指定一个可展开或折叠的代码块。
#endregion	它标识着 #region 块的结束。

#define 预处理器

define 预处理器指令创建符号常量。

define 允许您定义一个符号，这样，通过使用符号作为传递给 #if 指令的表达式，表达式将返回 true。它的语法如下：

```
#define symbol
```

下面的程序说明了这点：

```
#define PI
using System;
namespace PreprocessorDAppl
{
   class Program
   {
      static void Main(string[] args)
      {
         #if (PI)
            Console.WriteLine("PI is defined");
         #else
            Console.WriteLine("PI is not defined");
         #endif
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
PI is defined
```

条件指令

您可以使用 #if 指令来创建一个条件指令。条件指令用于测试符号是否为真。如果为真，编译器会执行 #if 和下一个指令之间的代码。

条件指令的语法：

```
#if symbol [operator symbol]...
```

其中，*symbol* 是要测试的符号名称。您也可以使用 true 和 false，或在符号前放置否定运算符。

运算符符号是用于评价符号的运算符。可以运算符可以是下列运算符之一：

- == (equality)
- != (inequality)
- && (and)

- || (or)

您也可以用括号把符号和运算符进行分组。条件指令用于在调试版本或编译指定配置时编译代码。一个以 **#if** 指令开始的条件指令，必须显示地以一个 **#endif** 指令终止。

下面的程序演示了条件指令的用法：

```
#define DEBUG
#define VC_V10
using System;
public class TestClass
{
   public static void Main()
   {

      #if (DEBUG && !VC_V10)
         Console.WriteLine("DEBUG is defined");
      #elif (!DEBUG && VC_V10)
         Console.WriteLine("VC_V10 is defined");
      #elif (DEBUG && VC_V10)
         Console.WriteLine("DEBUG and VC_V10 are defined");
      #else
         Console.WriteLine("DEBUG and VC_V10 are not defined");
      #endif
      Console.ReadKey();
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
DEBUG and VC_V10 are defined
```

C# 正则表达式

正则表达式 是一种匹配输入文本的模式。.Net 框架提供了允许这种匹配的正则表达式引擎。模式由一个或多个字符、运算符和结构组成。

定义正则表达式

下面列出了用于定义正则表达式的各种类别的字符、运算符和结构。

- 字符转义
- 字符类
- 定位点
- 分组构造
- 限定符
- 反向引用构造
- 备用构造
- 替换
- 杂项构造

字符转义

正则表达式中的反斜杠字符（\）指示其后跟的字符是特殊字符，或应按原义解释该字符。

下表列出了转义字符：

转义字符	描述	模式	匹配
\a	与报警 (bell) 符 \u0007 匹配。	\a	"Warning!" + '\u0007' 中的 "\u0007"
\b	在字符类中，与退格键 \u0008 匹配。	[\b]{3,}	"\b\b\b\b" 中的 "\b\b\b\b"
\t	与制表符 \u0009 匹配。	(\w+)\t	"Name\tAddr\t" 中的 "Name\t" 和 "Addr\t"
\r	与回车符 \u000D 匹配。（\r 与换行符 \n 不是等效的。）	\r\n(\w+)	"\r\Hello\nWorld." 中的 "\r\nHello"
\v	与垂直制表符 \u000B 匹配。	[\v]{2,}	"\v\v\v" 中的 "\v\v\v"
\f	与换页符 \u000C 匹配。	[\f]{2,}	"\f\f\f" 中的 "\f\f\f"
\n	与换行符 \u000A 匹配。	\r\n(\w+)	"\r\Hello\nWorld." 中的 "\r\nHello"
\e	与转义符 \u001B 匹配。	\e	"\x001B" 中的 "\x001B"
\ nnn	使用八进制表示形式指定一个字符（nnn 由二到三位数字组成）。	\w\040\w	"a bc d" 中的 "a b" 和 "c d"
\x nn	使用十六进制表示形式指定字符（nn 恰好由两位数字组成）。	\w\x20\w	"a bc d" 中的 "a b" 和 "c d"
\c X \c x	匹配 X 或 x 指定的 ASCII 控件字符，其中 X 或 x 是控件字符的字母。	\cC	"\x0003" 中的 "\x0003" (Ctrl-C)
\u nnnn	使用十六进制表示形式匹配一个 Unicode 字符（由 nnnn 表示的四位数）。	\w\u0020\w	"a bc d" 中的 "a b" 和 "c d"
****	在后面带有不识别的转义字符时，与该字符匹配。	\d+[+-x*]\d+\d+[+-x*\d+	"(2+2) *3*9" 中的 "2+2" 和 "3*9"

字符类

字符类与一组字符中的任何一个字符匹配。 下表列出了字符类：

字符类	描述	模式	匹配
[character_group]	匹配 character_group 中的任何单个字符。 默认情况下，匹配区分大小写。	[mn]	"mat" 中的 "m"， "moon" 中的 "m" 和 "n"
[^character_group]	非：与不在 character_group 中的任何单个字符匹配。 默认情况下，character_group 中的字符区分大小写。	[^aei]	"avail" 中的 "v" 和 "l"
[first - last]	字符范围：与从 first 到 last 的范围中的任何单个字符匹配。	(\w+)\t	"Name\tAddr\t" 中的 "Name\t" 和 "Addr\t"
.	通配符：与除 \n 之外的任何单个字符匹配。 若要匹配原意句点字符（. 或 \u002E），您必须在该字符前面加上转义符 (\.)。	a.e	"have" 中的 "ave"， "mate" 中的 "ate"
\p{ name }	与 *name* 指定的 Unicode 通用类别或命名块中的任何单个字符匹配。	\p{Lu}	"City Lights" 中的 "C" 和 "L"
\P{ name }	与不在 *name* 指定的 Unicode 通用类别或命名块中的任何单个字符匹配。	\P{Lu}	"City" 中的 "i"、 "t" 和 "y"
\w	与任何单词字符匹配。	\w	"Room#1" 中的 "R"、 "o"、 "m" 和 "1"
\W	与任何非单词字符匹配。	\W	"Room#1" 中的 "#"
\s	与任何空白字符匹配。	\w\s	"ID A1.3" 中的 "D "
\S	与任何非空白字符匹配。	\s\S	"int _*ctr*" 中的 " "
\d	与任何十进制数字匹配。	\d	"4 = IV" 中的 "4"
\D	匹配不是十进制数的任意字符。	\D	"4 = IV" 中的 " "、 "="、 " "、 "I" 和 "V"

定位点

定位点或原子零宽度断言会使匹配成功或失败，具体取决于字符串中的当前位置，但它们不会使引擎在字符串中前进或使用字符。 下表列出了定位点：

断言	描述	模式	匹配
^	匹配必须从字符串或一行的开头开始。	^\d{3}	"567-777-" 中的 "567"
$	匹配必须出现在字符串的末尾或出现在行或字符串末尾的 **\n** 之前。	-\d{4}$	"8-12-2012" 中的 "-2012"
\A	匹配必须出现在字符串的开头。	\A\w{3}	"Code-007-" 中的 "Code"
\Z	匹配必须出现在字符串的末尾或出现在字符串末尾的 **\n** 之前。	-\d{3}\Z	"Bond-901-007" 中的 "-007"
\z	匹配必须出现在字符串的末尾。	-\d{3}\z	"-901-333" 中的 "-333"
\G	匹配必须出现在上一个匹配结束的地方。	\G(\d)	"(1)(3)(5)7" 中的 "(1)"、 "(3)" 和 "(5)"
\b	匹配必须出现在 **\w**（字母数字）和 **\W**（非字母数字）字符之间的边界上。	\w	"Room#1" 中的 "R"、 "o"、 "m" 和 "1"
\B	匹配不得出现在 **\b** 边界上。	\Bend\w*\b	"end sends endure lender" 中的 "ends" 和 "ender"

分组构造

分组构造描述了正则表达式的子表达式，通常用于捕获输入字符串的子字符串。 下表列出了分组构造：

分组构造	描述	模式	匹配
(subexpression)	捕获匹配的子表达式并将其分配到一个从零开始的序号中。	(\w)\1	"deep" 中的 "ee"
(?< name >subexpression)	将匹配的子表达式捕获到一个命名组中。	(?< double>\w)\k< double>	"deep" 中的 "ee"
(?< name1 -name2 >subexpression)	定义平衡组定义。	(((?'Open'()\(\))+((?'Close-Open'))\(\))+)*(?(Open)(?!))$	"3+2^((1-3)(3-1))" 中的 "((1-3)(3-1))"
(?: subexpression)	定义非捕获组。	Write(?:Line)?	"Console.WriteLine()" 中的 "WriteLine"
(?imnsx-imnsx:subexpression)	应用或禁用 *subexpression* 中指定的选项。	A\d{2}(?i:\w+)\b	"A12xl A12XL a12xl" 中的 "A12xl" 和 "A12XL"
(?= subexpression)	零宽度正预测先行断言。	\w+(?=.)	"He is. The dog ran. The sun is out." 中的 "is"、 "ran" 和 "out"
(?! subexpression)	零宽度负预测先行断言。	\b(?!un)\w+\b	"unsure sure unity used" 中的 "sure" 和 "used"
(?< =subexpression)	零宽度正回顾后发断言。	(?< =19)\d{2}\b	"1851 1999 1950 1905 2003" 中的 "51" 和 "03"
(?< ! subexpression)	零宽度负回顾后发断言。	(?< !19)\d{2}\b	"end sends endure lender" 中的 "ends" 和 "ender"
(?> subexpression)	非回溯（也称为"贪婪"）子表达式。	[13579](?>A+B+)	"1ABB 3ABBC 5AB 5AC" 中的 "1ABB"、 "3ABB" 和 "5AB"

限定符

限定符指定在输入字符串中必须存在上一个元素（可以是字符、组或字符类）的多少个实例才能出现匹配项。 限定符包括下表中列出的语言元素。 下表列出了限定符：

限定符	描述	模式	匹配
*	匹配上一个元素零次或多次。	\d*.\d	".0"、 "19.9"、 "219.9"
+	匹配上一个元素一次或多次。	"be+"	"been" 中的 "bee"， "bent" 中的 "be"
?	匹配上一个元素零次或一次。	"rai?n"	"ran"、 "rain"
{ n }	匹配上一个元素恰好 n 次。	",\d{3}"	"1,043.6" 中的 ",043"， "9,876,543,210" 中的 ",876"、 ",543" 和 ",210"
{ n ,}	匹配上一个元素至少 n 次。	"\d{2,}"	"166"、 "29"、 "1930"
{ n , m }	匹配上一个元素至少 n 次， 但不多于 m 次。	"\d{3,5}"	"166"， "17668"， "193024" 中的 "19302"
***?**	匹配上一个元素零次或多次， 但次数尽可能少。	\d*?.\d	".0"、 "19.9"、 "219.9"
+?	匹配上一个元素一次或多次， 但次数尽可能少。	"be+?"	"been" 中的 "be"， "bent" 中的 "be"
??	匹配上一个元素零次或一次， 但次数尽可能少。	"rai??n"	"ran"、 "rain"
{ n }?	匹配前导元素恰好 n 次。	",\d{3}?"	"1,043.6" 中的 ",043"， "9,876,543,210" 中的 ",876"、 ",543" 和 ",210"
{ n ,}?	匹配上一个元素至少 n 次， 但次数尽可能少。	"\d{2,}?"	"166"、 "29" 和 "1930"
{ n , m }?	匹配上一个元素的次数介于 n 和 m 之间， 但次数尽可能少。	"\d{3,5}?"	"166"， "17668"， "193024" 中的 "193" 和 "024"

反向引用构造

反向引用允许在同一正则表达式中随后标识以前匹配的子表达式。 下表列出了反向引用构造：

反向引用构造	描述	模式	匹配
\ number	反向引用。 匹配编号子表达式的值。	(\w)\1	"seek" 中的 "ee"
\k< name >	命名反向引用。 匹配命名表达式的值。	(?< char>\w)\k< char>	"seek" 中的 "ee"

备用构造

备用构造用于修改正则表达式以启用 either/or 匹配。 下表列出了备用构造：

<table>
<tr><th>备用构造</th><th>描述</th><th>模式</th><th>匹配</th></tr>
<tr><td>|</td><td>匹配以竖线 (|) 字符分隔的任何一个元素。</td><td>th(e|is|at)</td><td>"this is the day. " 中的 "the" 和 "this"</td></tr>
<tr><td>(?(expression)yes | no)</td><td>如果正则表达式模式由 expression 匹配指定，则匹配 yes；否则匹配可选的 no 部分。 expression 被解释为零宽度断言。</td><td>(?(A)A\d{2}\b|\b\d{3}\b)</td><td>"A10 C103 910" 中的 "A10" 和 "910"</td></tr>
<tr><td>(?(name)yes | no)</td><td>如果 name 或已命名或已编号的捕获组具有匹配，则匹配 yes；否则匹配可选的 no。</td><td>(?< quoted>")?(?(quoted).+?"|\S+\s)</td><td>"Dogs.jpg "Yiska playing.jpg"" 中的 Dogs.jpg 和 "Yiska playing.jpg"</td></tr>
</table>

替换

替换是替换模式中使用的正则表达式。 下表列出了用于替换的字符：

字符	描述	模式	替换模式	输入字符串	结果字符串
$number	替换按组 *number* 匹配的子字符串。	\b(\w+)(\s)(\w+)\b	$3$2$1	"one two"	"two one"
${name}	替换按命名组 *name* 匹配的子字符串。	\b(?<word1>\w+)(\s)(?<word2>\w+)\b	${word2} ${word1}	"one two"	"two one"
$	替换字符"$"。	\b(\d+)\s?USD	$1	"103 USD"	"$103"
><	替换整个匹配项的一个副本。	($(*\d*(.+\d+)?){1})	**><	"$1.30"	**"$1.30"**
**	替换匹配前的输入字符串的所有文本。	B+	"AABBCC"	"AAAACC"	
/tr>	替换匹配后的输入字符串的所有文本。	B+	/tr>	"AABBCC"	"AACCCC"
$+	替换最后捕获的组。	B+(C+)	$+	"AABBCCDD"	AACCDD
$_	替换整个输入字符串。	B+	$_	"AABBCC"	"AAAABBCCCC

杂项构造 下表列出了各种杂项构造：

构造	描述	实例
(?imnsx-imnsx)	在模式中间对诸如不区分大小写这样的选项进行设置或禁用。	\bA(?i)b\w+\b 匹配 "ABA Able Act" 中的 "ABA" 和 "Able"
(?#comment)	内联注释。该注释在第一个右括号处终止。	\bA(?#Matches words starting with A)\w+\b
# [to end of line]	X 模式注释。 该注释以非转义的 # 开头，并继续到行的结尾。	(?x)\bA\w+\b#Matches words starting with A

Regex 类

Regex 类用于表示一个正则表达式。 下表列出了 Regex 类中一些常用的方法：

方法	描述
public bool IsMatch(string input)	指示 Regex 构造函数中指定的正则表达式是否在指定的输入字符串中找到匹配项。
public bool IsMatch(string input, int startat)	指示 Regex 构造函数中指定的正则表达式是否在指定的输入字符串中找到匹配项，从字符串中指定的开始位置开始。
public static bool IsMatch(string input, string pattern)	指示指定的正则表达式是否在指定的输入字符串中找到匹配项。
public MatchCollection Matches(string input)	在指定的输入字符串中搜索正则表达式的所有匹配项。
public string Replace(string input, string replacement)	在指定的输入字符串中，把所有匹配正则表达式模式的所有匹配的字符串替换为指定的替换字符串。
public string[] Split(string input)	把输入字符串分割为子字符串数组，根据在 Regex 构造函数中指定的正则表达式模式定义的位置进行分割。

如需了解 Regex 类的完整的属性列表，请参阅微软的 C# 文档。

实例 1

下面的实例匹配了以 'S' 开头的单词：

```
using System;
using System.Text.RegularExpressions;

namespace RegExApplication
{
   class Program
   {
      private static void showMatch(string text, string expr)
      {
         Console.WriteLine("The Expression: " + expr);
         MatchCollection mc = Regex.Matches(text, expr);
         foreach (Match m in mc)
         {
            Console.WriteLine(m);
         }
      }
      static void Main(string[] args)
      {
         string str = "A Thousand Splendid Suns";

         Console.WriteLine("Matching words that start with 'S': ");
         showMatch(str, @"\bS\S*");
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Matching words that start with 'S':
The Expression: \bS\S*
Splendid
Suns
```

实例 2

下面的实例匹配了以 'm' 开头以 'e' 结尾的单词：

```
using System;
using System.Text.RegularExpressions;

namespace RegExApplication
{
   class Program
   {
      private static void showMatch(string text, string expr)
      {
         Console.WriteLine("The Expression: " + expr);
         MatchCollection mc = Regex.Matches(text, expr);
         foreach (Match m in mc)
         {
            Console.WriteLine(m);
         }
      }
      static void Main(string[] args)
      {
         string str = "make maze and manage to measure it";

         Console.WriteLine("Matching words start with 'm' and ends with 'e':");
         showMatch(str, @"\bm\S*e\b");
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Matching words start with 'm' and ends with 'e':
The Expression: \bm\S*e\b
make
maze
manage
measure
```

实例 3

下面的实例替换掉多余的空格：

```
using System;
using System.Text.RegularExpressions;

namespace RegExApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         string input = "Hello   World   ";
         string pattern = "\\s+";
         string replacement = " ";
         Regex rgx = new Regex(pattern);
         string result = rgx.Replace(input, replacement);

         Console.WriteLine("Original String: {0}", input);
         Console.WriteLine("Replacement String: {0}", result);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Original String: Hello   World
Replacement String: Hello World
```

C# 异常处理

异常是在程序执行期间出现的问题。C# 中的异常是对程序运行时出现的特殊情况的一种响应，比如尝试除以零。

异常提供了一种把程序控制权从某个部分转移到另一个部分的方式。C# 异常处理时建立在四个关键词之上的：**try**、**catch**、**finally** 和 **throw**。

- **try**：一个 try 块标识了一个将被激活的特定的异常的代码块。后跟一个或多个 catch 块。
- **catch**：程序通过异常处理程序捕获异常。catch 关键字表示异常的捕获。
- **finally**：finally 块用于执行给定的语句，不管异常是否被抛出都会执行。例如，如果您打开一个文件，不管是否出现异常文件都要被关闭。
- **throw**：当问题出现时，程序抛出一个异常。使用 throw 关键字来完成。

语法

假设一个块将出现异常，一个方法使用 try 和 catch 关键字捕获异常。try/catch 块内的代码为受保护的代码，使用 try/catch 语法如下所示：

```
try
{
   // 引起异常的语句
}
catch( ExceptionName e1 )
{
   // 错误处理代码
}
catch( ExceptionName e2 )
{
   // 错误处理代码
}
catch( ExceptionName eN )
{
   // 错误处理代码
}
finally
{
   // 要执行的语句
}
```

您可以列出多个 catch 语句捕获不同类型的异常，以防 try 块在不同的情况下生成多个异常。

C# 中的异常类

C# 异常是使用类来表示的。C# 中的异常类主要是直接或间接地派生于 **System.Exception** 类。**System.ApplicationException** 和 **System.SystemException** 类是派生于 System.Exception 类的异常类。

System.ApplicationException 类支持由应用程序生成的异常。所以程序员定义的异常都应派生自该类。

System.SystemException 类是所有预定义的系统异常的基类。

下表列出了一些派生自 Sytem.SystemException 类的预定义的异常类：

异常类	描述
System.IO.IOException	处理 I/O 错误。
System.IndexOutOfRangeException	处理当方法指向超出范围的数组索引时生成的错误。
System.ArrayTypeMismatchException	处理当数组类型不匹配时生成的错误。
System.NullReferenceException	处理当依从一个空对象时生成的错误。
System.DivideByZeroException	处理当除以零时生成的错误。
System.InvalidCastException	处理在类型转换期间生成的错误。
System.OutOfMemoryException	处理空闲内存不足生成的错误。
System.StackOverflowException	处理栈溢出生成的错误。

异常处理

C# 以 try 和 catch 块的形式提供了一种结构化的异常处理方案。使用这些块，把核心程序语句与错误处理语句分离开。

这些错误处理块是使用 **try**、**catch** 和 **finally** 关键字实现的。下面是一个当除以零时抛出异常的实例：

```
using System;
namespace ErrorHandlingApplication
{
    class DivNumbers
    {
        int result;
        DivNumbers()
        {
            result = 0;
        }
        public void division(int num1, int num2)
        {
            try
            {
                result = num1 / num2;
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Exception caught: {0}", e);
            }
            finally
            {
                Console.WriteLine("Result: {0}", result);
            }

        }
        static void Main(string[] args)
        {
            DivNumbers d = new DivNumbers();
            d.division(25, 0);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

创建用户自定义异常

您也可以定义自己的异常。用户自定义的异常类是派生自 **ApplicationException** 类。下面的实例演示了这点：

```
using System;
namespace UserDefinedException
{
   class TestTemperature
   {
      static void Main(string[] args)
      {
         Temperature temp = new Temperature();
         try
         {
            temp.showTemp();
         }
         catch(TempIsZeroException e)
         {
            Console.WriteLine("TempIsZeroException: {0}", e.Message);
         }
         Console.ReadKey();
      }
   }
}
public class TempIsZeroException: ApplicationException
{
   public TempIsZeroException(string message): base(message)
   {
   }
}
public class Temperature
{
   int temperature = 0;
   public void showTemp()
   {
      if(temperature == 0)
      {
         throw (new TempIsZeroException("Zero Temperature found"));
      }
      else
      {
         Console.WriteLine("Temperature: {0}", temperature);
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
TempIsZeroException: Zero Temperature found
```

抛出对象

如果异常是直接或间接派生自 **System.Exception** 类，您可以抛出一个对象。您可以在 catch 块中使用 throw 语句来抛出当前的对象，如下所示：

```
Catch(Exception e)
{
   ...
   Throw e
}
```

C# 文件的输入与输出

一个 文件 是一个存储在磁盘中带有指定名称和目录路径的数据集合。当打开文件进行读写时，它变成一个 流。

从根本上说，流是通过通信路径传递的字节序列。有两个主要的流：输入流 和 输出流。输入流用于从文件读取数据（读操作），输出流用于向文件写入数据（写操作）。

C# I/O 类

System.IO 命名空间有各种不同的类，用于执行各种文件操作，如创建和删除文件、读取或写入文件，关闭文件等。

下表列出了一些 System.IO 命名空间中常用的非抽象类：

I/O 类	描述
BinaryReader	从二进制流读取原始数据。
BinaryWriter	以二进制格式写入原始数据。
BufferedStream	字节流的临时存储。
Directory	有助于操作目录结构。
DirectoryInfo	用于对目录执行操作。
DriveInfo	提供驱动器的信息。
File	有助于处理文件。
FileInfo	用于对文件执行操作。
FileStream	用于文件中任何位置的读写。
MemoryStream	用于随机访问存储在内存中的数据流。
Path	对路径信息执行操作。
StreamReader	用于从字节流中读取字符。
StreamWriter	用于向一个流中写入字符。
StringReader	用于读取字符串缓冲区。
StringWriter	用于写入字符串缓冲区。

FileStream 类

System.IO 命名空间中的 **FileStream** 类有助于文件的读写与关闭。该类派生自抽象类 Stream。

您需要创建一个 **FileStream** 对象来创建一个新的文件，或打开一个已有的文件。创建 **FileStream** 对象的语法如下：

```
FileStream <object_name> = new FileStream( <file_name>,
<FileMode Enumerator>, <FileAccess Enumerator>, <FileShare Enumerator>);
```

例如，创建一个 FileStream 对象 **F** 来读取名为 **sample.txt** 的文件：

```
FileStream F = new FileStream("sample.txt", FileMode.Open, FileAccess.Read, FileShare.Rea
```

参数	描述
FileMode	**FileMode** 枚举定义了各种打开文件的方法。FileMode 枚举的成员有：**Append**：打开一个已有的文件，并将光标放置在文件的末尾。如果文件不存在，则创建文件。 ***Create***：创建一个新的文件。如果文件已存在，则删除旧文件，然后创建新文件。 ***CreateNew***：指定操作系统应创建一个新的文件。如果文件已存在，则抛出异常。 ***Open***：打开一个已有的文件。如果文件不存在，则抛出异常。 ***OpenOrCreate***：指定操作系统应打开一个已有的文件。如果文件不存在，则用指定的名称创建一个新的文件打开。 **Truncate*：打开一个已有的文件，文件一旦打开，就将被截断为零字节大小。然后我们可以向文件写入全新的数据，但是保留文件的初始创建日期。如果文件不存在，则抛出异常。
FileAccess	**FileAccess** 枚举的成员有：**Read**、**ReadWrite** 和 **Write**。
FileShare	**FileShare** 枚举的成员有： **Inheritable**：允许文件句柄可由子进程继承。Win32 不直接支持此功能。 **None**：谢绝共享当前文件。文件关闭前，打开该文件的任何请求（由此进程或另一进程发出的请求）都将失败。 **Read**：允许随后打开文件读取。如果未指定此标志，则文件关闭前，任何打开该文件以进行读取的请求（由此进程或另一进程发出的请求）都将失败。但是，即使指定了此标志，仍可能需要附加权限才能够访问该文件。 **ReadWrite**：允许随后打开文件读取或写入。如果未指定此标志，则文件关闭前，任何打开该文件以进行读取或写入的请求（由此进程或另一进程发出）都将失败。但是，即使指定了此标志，仍可能需要附加权限才能够访问该文件。 **Write**：允许随后打开文件写入。如果未指定此标志，则文件关闭前，任何打开该文件以进行写入的请求（由此进程或另一进过程发出的请求）都将失败。但是，即使指定了此标志，仍可能需要附加权限才能够访问该文件。 **Delete**：允许随后删除文件。

实例

下面的程序演示了 **FileStream** 类的用法：

```
using System;
using System.IO;

namespace FileIOApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            FileStream F = new FileStream("test.dat",
            FileMode.OpenOrCreate, FileAccess.ReadWrite);

            for (int i = 1; i <= 20; i++)
            {
                F.WriteByte((byte)i);
            }

            F.Position = 0;

            for (int i = 0; i <= 20; i++)
            {
                Console.Write(F.ReadByte() + " ");
            }
            F.Close();
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

C# 高级文件操作

上面的实例演示了 C# 中简单的文件操作。但是，要充分利用 C# System.IO 类的强大功能，您需要知道这些类常用的属性和方法。

在下面的章节中，我们将讨论这些类和它们执行的操作。请单击链接详细了解各个部分的知识：

主题	描述
文本文件的读写	它涉及到文本文件的读写。**StreamReader** 和 **StreamWriter** 类有助于完成文本文件的读写。
二进制文件的读写	它涉及到二进制文件的读写。**BinaryReader** 和 **BinaryWriter** 类有助于完成二进制文件的读写。
Windows 文件系统的操作	它让 C# 程序员能够浏览并定位 Windows 文件和目录。

C# 高级

C# 特性（Attribute）

特性（**Attribute**）是用于在运行时传递程序中各种元素（比如类、方法、结构、枚举、组件等）的行为信息的声明性标签。您可以通过使用特性向程序添加声明性信息。一个声明性标签是通过放置在它所应用的元素前面的方括号（[ ]）来描述的。

特性（Attribute）用于添加元数据，如编译器指令和注释、描述、方法、类等其他信息。.Net 框架提供了两种类型的特性：预定义特性和自定义特性。

规定特性（Attribute）

规定特性（Attribute）的语法如下：

```
[attribute(positional_parameters, name_parameter = value, ...)]
element
```

特性（Attribute）的名称和值是在方括号内规定的，放置在它所应用的元素之前。positional_parameters 规定必需的信息，name_parameter 规定可选的信息。

预定义特性（Attribute）

.Net 框架提供了三种预定义特性：

- AttributeUsage
- Conditional
- Obsolete

AttributeUsage

预定义特性 **AttributeUsage** 描述了如何使用一个自定义特性类。它规定了特性可应用到的项目的类型。

规定该特性的语法如下：

```
[AttributeUsage(
   validon,
   AllowMultiple=allowmultiple,
   Inherited=inherited
)]
```

其中：

- 参数 validon 规定特性可被放置的语言元素。它是枚举器 *AttributeTargets* 的值的组合。默认值是 *AttributeTargets.All*。
- 参数 *allowmultiple*（可选的）为该特性的 *AllowMultiple* 属性（property）提供一个布尔值。如果为 true，则该特性是多用的。默认值是 false（单用的）。
- 参数 *inherited*（可选的）为该特性的 *Inherited* 属性（property）提供一个布尔值。如果为 true，则该特性可被派生类继承。默认值是 false（不被继承）。

例如：

```
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Feild |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]
```

Conditional

这个预定义特性标记了一个条件方法，其执行依赖于它顶的预处理标识符。

它会引起方法调用的条件编译，取决于指定的值，比如 **Debug** 或 **Trace**。例如，当调试代码时显示变量的值。

规定该特性的语法如下：

```
[Conditional(
   conditionalSymbol
)]
```

例如：

```
[Conditional("DEBUG")]
```

下面的实例演示了该特性：

```
#define DEBUG
using System;
using System.Diagnostics;
public class Myclass
{
    [Conditional("DEBUG")]
    public static void Message(string msg)
    {
        Console.WriteLine(msg);
    }
}
class Test
{
    static void function1()
    {
        Myclass.Message("In Function 1.");
        function2();
    }
    static void function2()
    {
        Myclass.Message("In Function 2.");
    }
    public static void Main()
    {
        Myclass.Message("In Main function.");
        function1();
        Console.ReadKey();
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
In Main function
In Function 1
In Function 2
```

Obsolete

这个预定义特性标记了不应被使用的程序实体。它可以让您通知编译器丢弃某个特定的目标元素。例如，当一个新方法被用在一个类中，但是您仍然想要保持类中的旧方法，您可以通过显示一个应该使用新方法，而不是旧方法的消息，来把它标记为 obsolete（过时的）。

规定该特性的语法如下：

```
[Obsolete(
   message
)]
[Obsolete(
   message,
   iserror
)]
```

其中：

- 参数 *message*，是一个字符串，描述项目为什么过时的原因以及该替代使用什么。
- 参数 *iserror*，是一个布尔值。如果该值为 true，编译器应把该项目的使用当作一个错误。默认值是 false（编译器生成一个警告）。

下面的实例演示了该特性：

```
using System;
public class MyClass
{
   [Obsolete("Don't use OldMethod, use NewMethod instead", true)]
   static void OldMethod()
   {
      Console.WriteLine("It is the old method");
   }
   static void NewMethod()
   {
      Console.WriteLine("It is the new method");
   }
   public static void Main()
   {
      OldMethod();
   }
}
```

当您尝试编译该程序时，编译器会给出一个错误消息说明：

```
Don't use OldMethod, use NewMethod instead
```

创建自定义特性（Attribute）

.Net 框架允许创建自定义特性，用于存储声明性的信息，且可在运行时被检索。该信息根据设计标准和应用程序需要，可与任何目标元素相关。

创建并使用自定义特性包含四个步骤：

- 声明自定义特性
- 构建自定义特性
- 在目标程序元素上应用自定义特性
- 通过反射访问特性

最后一个步骤包含编写一个简单的程序来读取元数据以便查找各种符号。元数据是用于描述其他数据的数据和信息。该程序应使用反射来在运行时访问特性。我们将在下一章详细讨论这点。

声明自定义特性

一个新的自定义特性应派生自 **System.Attribute** 类。例如：

```
// 一个自定义特性 BugFix 被赋给类及其成员
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]

public class DeBugInfo : System.Attribute
```

在上面的代码中，我们已经声明了一个名为 *DeBugInfo* 的自定义特性。

构建自定义特性

让我们构建一个名为 *DeBugInfo* 的自定义特性，该特性将存储调试程序获得的信息。它存储下面的信息：

- bug 的代码编号
- 辨认该 bug 的开发人员名字
- 最后一次审查该代码的日期
- 一个存储了开发人员标记的字符串消息

我们的 *DeBugInfo* 类将带有三个用于存储前三个信息的私有属性（property）和一个用于存储消息的公有属性（property）。所以 bug 编号、开发人员名字和审查日期将是 DeBugInfo 类的必需的定位（ positional）参数，消息将是一个可选的命名（named）参数。

每个特性必须至少有一个构造函数。必需的定位（ positional）参数应通过构造函数传递。下面的代码演示了 *DeBugInfo* 类：

```
// 一个自定义特性 BugFix 被赋给类及其成员
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]

public class DeBugInfo : System.Attribute
{
  private int bugNo;
  private string developer;
  private string lastReview;
  public string message;

  public DeBugInfo(int bg, string dev, string d)
  {
      this.bugNo = bg;
      this.developer = dev;
      this.lastReview = d;
  }

  public int BugNo
  {
      get
      {
          return bugNo;
      }
  }
  public string Developer
  {
      get
      {
          return developer;
      }
  }
  public string LastReview
  {
      get
      {
          return lastReview;
      }
  }
  public string Message
  {
      get
      {
          return message;
      }
      set
      {
          message = value;
      }
  }
}
```

应用自定义特性

通过把特性放置在紧接着它的目标之前，来应用该特性：

```
[DeBugInfo(45, "Zara Ali", "12/8/2012", Message = "Return type mismatch")]
[DeBugInfo(49, "Nuha Ali", "10/10/2012", Message = "Unused variable")]
class Rectangle
{
  // 成员变量
  protected double length;
  protected double width;
  public Rectangle(double l, double w)
  {
      length = l;
      width = w;
  }
  [DeBugInfo(55, "Zara Ali", "19/10/2012",
  Message = "Return type mismatch")]
  public double GetArea()
  {
      return length * width;
  }
  [DeBugInfo(56, "Zara Ali", "19/10/2012")]
  public void Display()
  {
      Console.WriteLine("Length: {0}", length);
      Console.WriteLine("Width: {0}", width);
      Console.WriteLine("Area: {0}", GetArea());
  }
}
```

在下一章中，我们将使用 Reflection 类对象来检索这些信息。

C# 反射（Reflection）

反射（**Reflection**） 对象用于在运行时获取类型信息。该类位于 **System.Reflection** 命名空间中，可访问一个正在运行的程序的元数据。

System.Reflection 命名空间包含了允许您获取有关应用程序信息及向应用程序动态添加类型、值和对象的类。

反射（Reflection）的用途

反射（Reflection）有下列用途：

- 它允许在运行时查看属性（attribute）信息。
- 它允许审查集合中的各种类型，以及实例化这些类型。
- 它允许延迟绑定的方法和属性（property）。
- 它允许在运行时创建新类型，然后使用这些类型执行一些任务。

查看元数据

我们已经在上面的章节中提到过，使用反射（Reflection）可以查看属性（attribute）信息。

System.Reflection 类的 **MemberInfo** 对象需要被初始化，用于发现与类相关的属性（attribute）。为了做到这点，您可以定义目标类的一个对象，如下：

```
System.Reflection.MemberInfo info = typeof(MyClass);
```

下面的程序演示了这点：

```
using System;

[AttributeUsage(AttributeTargets.All)]
public class HelpAttribute : System.Attribute
{
   public readonly string Url;

   public string Topic  // Topic 是一个命名（named）参数
   {
      get
      {
         return topic;
      }
      set
      {

         topic = value;
      }
   }

   public HelpAttribute(string url)  // url 是一个定位（positional）参数
   {
      this.Url = url;
   }

   private string topic;
}
[HelpAttribute("Information on the class MyClass")]
class MyClass
{
}

namespace AttributeAppl
{
   class Program
   {
      static void Main(string[] args)
      {
         System.Reflection.MemberInfo info = typeof(MyClass);
         object[] attributes = info.GetCustomAttributes(true);
         for (int i = 0; i < attributes.Length; i++)
         {
            System.Console.WriteLine(attributes[i]);
         }
         Console.ReadKey();

      }
   }
}
```

当上面的代码被编译和执行时，它会显示附加到类 *MyClass* 上的自定义属性：

```
HelpAttribute
```

实例

在本实例中，我们将使用在上一章中创建的 *DeBugInfo* 属性，并使用反射（Reflection）来读取 *Rectangle* 类中的元数据。

```
using System;
using System.Reflection;
```

```
namespace BugFixApplication
{
   // 一个自定义属性 BugFix 被赋给类及其成员
   [AttributeUsage(AttributeTargets.Class |
   AttributeTargets.Constructor |
   AttributeTargets.Field |
   AttributeTargets.Method |
   AttributeTargets.Property,
   AllowMultiple = true)]

   public class DeBugInfo : System.Attribute
   {
      private int bugNo;
      private string developer;
      private string lastReview;
      public string message;

      public DeBugInfo(int bg, string dev, string d)
      {
         this.bugNo = bg;
         this.developer = dev;
         this.lastReview = d;
      }

      public int BugNo
      {
         get
         {
            return bugNo;
         }
      }
      public string Developer
      {
         get
         {
            return developer;
         }
      }
      public string LastReview
      {
         get
         {
            return lastReview;
         }
      }
      public string Message
      {
         get
         {
            return message;
         }
         set
         {
            message = value;
         }
      }
   }
   [DeBugInfo(45, "Zara Ali", "12/8/2012",
    Message = "Return type mismatch")]
   [DeBugInfo(49, "Nuha Ali", "10/10/2012",
    Message = "Unused variable")]
   class Rectangle
   {
      // 成员变量
      protected double length;
      protected double width;
      public Rectangle(double l, double w)
      {
         length = l;
         width = w;
      }
      [DeBugInfo(55, "Zara Ali", "19/10/2012",
```

```
         Message = "Return type mismatch")]
        public double GetArea()
        {
           return length * width;
        }
        [DeBugInfo(56, "Zara Ali", "19/10/2012")]
        public void Display()
        {
           Console.WriteLine("Length: {0}", length);
           Console.WriteLine("Width: {0}", width);
           Console.WriteLine("Area: {0}", GetArea());
        }
     }//end class Rectangle

     class ExecuteRectangle
     {
        static void Main(string[] args)
        {
           Rectangle r = new Rectangle(4.5, 7.5);
           r.Display();
           Type type = typeof(Rectangle);
           // 遍历 Rectangle 类的属性
           foreach (Object attributes in type.GetCustomAttributes(false))
           {
              DeBugInfo dbi = (DeBugInfo)attributes;
              if (null != dbi)
              {
                 Console.WriteLine("Bug no: {0}", dbi.BugNo);
                 Console.WriteLine("Developer: {0}", dbi.Developer);
                 Console.WriteLine("Last Reviewed: {0}",
                      dbi.LastReview);
                 Console.WriteLine("Remarks: {0}", dbi.Message);
              }
           }

           // 遍历方法属性
           foreach (MethodInfo m in type.GetMethods())
           {
              foreach (Attribute a in m.GetCustomAttributes(true))
              {
                 DeBugInfo dbi = (DeBugInfo)a;
                 if (null != dbi)
                 {
                    Console.WriteLine("Bug no: {0}, for Method: {1}",
                          dbi.BugNo, m.Name);
                    Console.WriteLine("Developer: {0}", dbi.Developer);
                    Console.WriteLine("Last Reviewed: {0}",
                          dbi.LastReview);
                    Console.WriteLine("Remarks: {0}", dbi.Message);
                 }
              }
           }
           Console.ReadLine();
        }
     }
  }
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Length: 4.5
Width: 7.5
Area: 33.75
Bug No: 49
Developer: Nuha Ali
Last Reviewed: 10/10/2012
Remarks: Unused variable
Bug No: 45
Developer: Zara Ali
Last Reviewed: 12/8/2012
Remarks: Return type mismatch
Bug No: 55, for Method: GetArea
Developer: Zara Ali
Last Reviewed: 19/10/2012
Remarks: Return type mismatch
Bug No: 56, for Method: Display
Developer: Zara Ali
Last Reviewed: 19/10/2012
Remarks:
```

C# 属性（Property）

属性（**Property**） 是类（class）、结构（structure）和接口（interface）的命名（named）成员。类或结构中的成员变量或方法称为 域（**Field**）。属性（Property）是域（Field）的扩展，且可使用相同的语法来访问。它们使用 访问器（**accessors**） 让私有域的值可被读写或操作。

属性（Property）不会确定存储位置。相反，它们具有可读写或计算它们值的 访问器（**accessors**）。

例如，有一个名为 Student 的类，带有 age、name 和 code 的私有域。我们不能在类的范围以外直接访问这些域，但是我们可以拥有访问这些私有域的属性。

访问器（Accessors）

属性（Property）的访问器（**accessor**）包含有助于获取（读取或计算）或设置（写入）属性的可执行语句。访问器（accessor）声明可包含一个 get 访问器、一个 set 访问器，或者同时包含二者。例如：

```
// 声明类型为 string 的 Code 属性
public string Code
{
   get
   {
      return code;
   }
   set
   {
      code = value;
   }
}

// 声明类型为 string 的 Name 属性
public string Name
{
   get
   {
     return name;
   }
   set
   {
     name = value;
   }
}

// 声明类型为 int 的 Age 属性
public int Age
{
   get
   {
      return age;
   }
   set
   {
      age = value;
   }
}
```

实例

下面的实例演示了属性（Property）的用法：

```
using System;
namespace tutorialspoint
{
   class Student
   {

      private string code = "N.A";
      private string name = "not known";
      private int age = 0;

      // 声明类型为 string 的 Code 属性
      public string Code
      {
         get
         {
            return code;
         }
         set
         {
            code = value;
         }
      }
```

```
        // 声明类型为 string 的 Name 属性
        public string Name
        {
           get
           {
              return name;
           }
           set
           {
              name = value;
           }
        }

        // 声明类型为 int 的 Age 属性
        public int Age
        {
           get
           {
              return age;
           }
           set
           {
              age = value;
           }
        }
        public override string ToString()
        {
           return "Code = " + Code +", Name = " + Name + ", Age = " + Age;
        }
      }
      class ExampleDemo
      {
        public static void Main()
        {
           // 创建一个新的 Student 对象
           Student s = new Student();

           // 设置 student 的 code、name 和 age
           s.Code = "001";
           s.Name = "Zara";
           s.Age = 9;
           Console.WriteLine("Student Info: {0}", s);
           // 增加年龄
           s.Age += 1;
           Console.WriteLine("Student Info: {0}", s);
           Console.ReadKey();
         }
     }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Student Info: Code = 001, Name = Zara, Age = 9
Student Info: Code = 001, Name = Zara, Age = 10
```

抽象属性（Abstract Properties）

抽象类可拥有抽象属性，这些属性应在派生类中被实现。下面的程序说明了这点：

```
using System;
namespace tutorialspoint
{
```

```
    public abstract class Person
    {
       public abstract string Name
       {
          get;
          set;
       }
       public abstract int Age
       {
          get;
          set;
       }
    }
    class Student : Person
    {

       private string code = "N.A";
       private string name = "N.A";
       private int age = 0;

       // 声明类型为 string 的 Code 属性
       public string Code
       {
          get
          {
             return code;
          }
          set
          {
             code = value;
          }
       }

       // 声明类型为 string 的 Name 属性
       public override string Name
       {
          get
          {
             return name;
          }
          set
          {
             name = value;
          }
       }

       // 声明类型为 int 的 Age 属性
       public override int Age
       {
          get
          {
             return age;
          }
          set
          {
             age = value;
          }
       }
       public override string ToString()
       {
          return "Code = " + Code +", Name = " + Name + ", Age = " + Age;
       }
    }
    class ExampleDemo
    {
       public static void Main()
       {
          // 创建一个新的 Student 对象
          Student s = new Student();

          // 设置 student 的 code、name 和 age
          s.Code = "001";
```

```
            s.Name = "Zara";
            s.Age = 9;
            Console.WriteLine("Student Info:- {0}", s);
            // 增加年龄
            s.Age += 1;
            Console.WriteLine("Student Info:- {0}", s);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Student Info: Code = 001, Name = Zara, Age = 9
Student Info: Code = 001, Name = Zara, Age = 10
```

C# 索引器（Indexer）

索引器（**Indexer**） 允许一个对象可以像数组一样被索引。当您为类定义一个索引器时，该类的行为就会像一个 虚拟数组（**virtual array**） 一样。您可以使用数组访问运算符（[ ]）来访问该类的实例。

语法

一维索引器的语法如下：

```
element-type this[int index]
{
   // get 访问器
   get
   {
      // 返回 index 指定的值
   }

   // set 访问器
   set
   {
      // 设置 index 指定的值
   }
}
```

索引器（Indexer）的用途

索引器的行为的声明在某种程度上类似于属性（property）。就像属性（property），您可使用 **get** 和 **set** 访问器来定义索引器。但是，属性返回或设置一个特定的数据成员，而索引器返回或设置对象实例的一个特定值。换句话说，它把实例数据分为更小的部分，并索引每个部分，获取或设置每个部分。

定义一个属性（property）包括提供属性名称。索引器定义的时候不带有名称，但带有 **this** 关键字，它指向对象实例。下面的实例演示了这个概念：

```
using System;
namespace IndexerApplication
{
   class IndexedNames
   {
      private string[] namelist = new string[size];
      static public int size = 10;
      public IndexedNames()
      {
         for (int i = 0; i < size; i++)
         namelist[i] = "N. A.";
      }
      public string this[int index]
      {
         get
         {
            string tmp;

            if( index >= 0 && index <= size-1 )
            {
               tmp = namelist[index];
            }
            else
            {
               tmp = "";
            }

            return ( tmp );
         }
         set
         {
            if( index >= 0 && index <= size-1 )
            {
               namelist[index] = value;
            }
         }
      }

      static void Main(string[] args)
      {
         IndexedNames names = new IndexedNames();
         names[0] = "Zara";
         names[1] = "Riz";
         names[2] = "Nuha";
         names[3] = "Asif";
         names[4] = "Davinder";
         names[5] = "Sunil";
         names[6] = "Rubic";
         for ( int i = 0; i < IndexedNames.size; i++ )
         {
            Console.WriteLine(names[i]);
         }
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Zara
Riz
Nuha
Asif
Davinder
Sunil
Rubic
N. A.
N. A.
N. A.
```

重载索引器（Indexer）

索引器（Indexer）可被重载。索引器声明的时候也可带有多个参数，且每个参数可以是不同的类型。没有必要让索引器必须是整型的。C# 允许索引器可以是其他类型，例如，字符串类型。

下面的实例演示了重载索引器：

```
using System;
namespace IndexerApplication
{
   class IndexedNames
   {
      private string[] namelist = new string[size];
      static public int size = 10;
      public IndexedNames()
      {
         for (int i = 0; i < size; i++)
         {
          namelist[i] = "N. A.";
         }
      }
      public string this[int index]
      {
         get
         {
            string tmp;

            if( index >= 0 && index <= size-1 )
            {
               tmp = namelist[index];
            }
            else
            {
               tmp = "";
            }

            return ( tmp );
         }
         set
         {
            if( index >= 0 && index <= size-1 )
            {
               namelist[index] = value;
            }
         }
      }
      public int this[string name]
      {
         get
         {
            int index = 0;
```

```
            while(index < size)
            {
               if (namelist[index] == name)
               {
                return index;
               }
               index++;
            }
            return index;
         }

      }

      static void Main(string[] args)
      {
         IndexedNames names = new IndexedNames();
         names[0] = "Zara";
         names[1] = "Riz";
         names[2] = "Nuha";
         names[3] = "Asif";
         names[4] = "Davinder";
         names[5] = "Sunil";
         names[6] = "Rubic";
         // 使用带有 int 参数的第一个索引器
         for (int i = 0; i < IndexedNames.size; i++)
         {
            Console.WriteLine(names[i]);
         }
         // 使用带有 string 参数的第二个索引器
         Console.WriteLine(names["Nuha"]);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Zara
Riz
Nuha
Asif
Davinder
Sunil
Rubic
N. A.
N. A.
N. A.
2
```

C# 委托（Delegate）

C# 中的委托（Delegate）类似于 C 或 C++ 中函数的指针。委托（**Delegate**） 是存有对某个方法的引用的一种引用类型变量。引用可在运行时被改变。

委托（Delegate）特别用于实现事件和回调方法。所有的委托（Delegate）都派生自 **System.Delegate** 类。

声明委托（Delegate）

委托声明决定了可由该委托引用的方法。委托可指向一个与其具有相同标签的方法。

例如，假设有一个委托：

```
public delegate int MyDelegate (string s);
```

上面的委托可被用于引用任何一个带有一个单一的 *string* 参数的方法，并返回一个 *int* 类型变量。

声明委托的语法如下：

```
delegate <return type> <delegate-name> <parameter list>
```

实例化委托（Delegate）

一旦声明了委托类型，委托对象必须使用 **new** 关键字来创建，且与一个特定的方法有关。当创建委托时，传递到 **new** 语句的参数就像方法调用一样书写，但是不带有参数。例如：

```
public delegate void printString(string s);
...
printString ps1 = new printString(WriteToScreen);
printString ps2 = new printString(WriteToFile);
```

下面的实例演示了委托的声明、实例化和使用，该委托可用于引用带有一个整型参数的方法，并返回一个整型值。

```
using System;

delegate int NumberChanger(int n);
namespace DelegateAppl
{
   class TestDelegate
   {
      static int num = 10;
      public static int AddNum(int p)
      {
         num += p;
         return num;
      }

      public static int MultNum(int q)
      {
         num *= q;
         return num;
      }
      public static int getNum()
      {
         return num;
      }

      static void Main(string[] args)
      {
         // 创建委托实例
         NumberChanger nc1 = new NumberChanger(AddNum);
         NumberChanger nc2 = new NumberChanger(MultNum);
         // 使用委托对象调用方法
         nc1(25);
         Console.WriteLine("Value of Num: {0}", getNum());
         nc2(5);
         Console.WriteLine("Value of Num: {0}", getNum());
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Value of Num: 35
Value of Num: 175
```

委托的多播（**Multicasting of a Delegate**）

委托对象可使用 "+" 运算符进行合并。一个合并委托调用它所合并的两个委托。只有相同类型的委托可被合并。"-" 运算符可用于从合并的委托中移除组件委托。

使用委托的这个有用的特点，您可以创建一个委托被调用时要调用的方法的调用列表。这被称为委托的 多播（**multicasting**），也叫组播。下面的程序演示了委托的多播：

```
using System;

delegate int NumberChanger(int n);
namespace DelegateAppl
{
   class TestDelegate
   {
      static int num = 10;
      public static int AddNum(int p)
      {
         num += p;
         return num;
      }

      public static int MultNum(int q)
      {
         num *= q;
         return num;
      }
      public static int getNum()
      {
         return num;
      }

      static void Main(string[] args)
      {
         // 创建委托实例
         NumberChanger nc;
         NumberChanger nc1 = new NumberChanger(AddNum);
         NumberChanger nc2 = new NumberChanger(MultNum);
         nc = nc1;
         nc += nc2;
         // 调用多播
         nc(5);
         Console.WriteLine("Value of Num: {0}", getNum());
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Value of Num: 75
```

委托（Delegate）的用途

下面的实例演示了委托的用法。委托 *printString* 可用于引用带有一个字符串作为输入的方法，并不返回任何东西。

我们使用这个委托来调用两个方法，第一个把字符串打印到控制台，第二个把字符串打印到文件：

```
using System;
using System.IO;

namespace DelegateAppl
{
   class PrintString
   {
      static FileStream fs;
      static StreamWriter sw;
      // 委托声明
      public delegate void printString(string s);

      // 该方法打印到控制台
      public static void WriteToScreen(string str)
      {
         Console.WriteLine("The String is: {0}", str);
      }
      // 该方法打印到文件
      public static void WriteToFile(string s)
      {
         fs = new FileStream("c:\\message.txt",
         FileMode.Append, FileAccess.Write);
         sw = new StreamWriter(fs);
         sw.WriteLine(s);
         sw.Flush();
         sw.Close();
         fs.Close();
      }
      // 该方法把委托作为参数，并使用它调用方法
      public static void sendString(printString ps)
      {
         ps("Hello World");
      }
      static void Main(string[] args)
      {
         printString ps1 = new printString(WriteToScreen);
         printString ps2 = new printString(WriteToFile);
         sendString(ps1);
         sendString(ps2);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
The String is: Hello World
```

C# 事件（Event）

事件（**Event**） 基本上说是一个用户操作，如按键、点击、鼠标移动等等，或者是一些出现，如系统生成的通知。应用程序需要在事件发生时响应事件。例如，中断。事件是用于进程间通信。

通过事件使用委托

事件在类中声明且生成，且通过使用同一个类或其他类中的委托与事件处理程序关联。包含事件的类用于发布事件。这被称为 发布器（**publisher**） 类。其他接受该事件的类被称为 订阅器（**subscriber**） 类。事件使用 发布-订阅（**publisher-subscriber**） 模型。

发布器（**publisher**） 是一个包含事件和委托定义的对象。事件和委托之间的联系也定义在这个对象中。发布器（publisher）类的对象调用这个事件，并通知其他的对象。

订阅器（**subscriber**） 是一个接受事件并提供事件处理程序的对象。在发布器（publisher）类中的委托调用订阅器（subscriber）类中的方法（事件处理程序）。

声明事件（Event）

在类的内部声明事件，首先必须声明该事件的委托类型。例如：

```
public delegate void BoilerLogHandler(string status);
```

然后，声明事件本身，使用 **event** 关键字：

```
// 基于上面的委托定义事件
public event BoilerLogHandler BoilerEventLog;
```

上面的代码定义了一个名为 *BoilerLogHandler* 的委托和一个名为 *BoilerEventLog* 的事件，该事件在生成的时候会调用委托。

实例 1

```
using System;
namespace SimpleEvent
{
   using System;

   public class EventTest
   {
      private int value;

      public delegate void NumManipulationHandler();

      public event NumManipulationHandler ChangeNum;

      protected virtual void OnNumChanged()
      {
         if (ChangeNum != null)
         {
            ChangeNum();
         }
         else
         {
            Console.WriteLine("Event fired!");
         }

      }
      public EventTest(int n )
      {
         SetValue(n);
      }
      public void SetValue(int n)
      {
         if (value != n)
         {
            value = n;
            OnNumChanged();
         }
      }
   }
   public class MainClass
   {
      public static void Main()
      {
         EventTest e = new EventTest(5);
         e.SetValue(7);
         e.SetValue(11);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Event Fired!
Event Fired!
Event Fired!
```

实例 2

本实例提供一个简单的用于热水锅炉系统故障排除的应用程序。当维修工程师检查锅炉时，锅炉的温度和压力会随着维修工程师的备注自动记录到日志文件中。

```
using System;
using System.IO;

namespace BoilerEventAppl
{

   // boiler 类
   class Boiler
   {
      private int temp;
      private int pressure;
      public Boiler(int t, int p)
      {
         temp = t;
         pressure = p;
      }

      public int getTemp()
      {
         return temp;
      }
      public int getPressure()
      {
         return pressure;
      }
   }
   // 事件发布器
   class DelegateBoilerEvent
   {
      public delegate void BoilerLogHandler(string status);

      // 基于上面的委托定义事件
      public event BoilerLogHandler BoilerEventLog;

      public void LogProcess()
      {
         string remarks = "O. K";
         Boiler b = new Boiler(100, 12);
         int t = b.getTemp();
         int p = b.getPressure();
         if(t > 150 || t < 80 || p < 12 || p > 15)
         {
            remarks = "Need Maintenance";
         }
         OnBoilerEventLog("Logging Info:\n");
         OnBoilerEventLog("Temparature " + t + "\nPressure: " + p);
         OnBoilerEventLog("\nMessage: " + remarks);
      }

      protected void OnBoilerEventLog(string message)
      {
         if (BoilerEventLog != null)
         {
            BoilerEventLog(message);
         }
      }
   }
   // 该类保留写入日志文件的条款
   class BoilerInfoLogger
   {
      FileStream fs;
      StreamWriter sw;
      public BoilerInfoLogger(string filename)
      {
         fs = new FileStream(filename, FileMode.Append, FileAccess.Write);
         sw = new StreamWriter(fs);
      }
      public void Logger(string info)
      {
         sw.WriteLine(info);
      }
```

```
        public void Close()
        {
            sw.Close();
            fs.Close();
        }
    }
    // 事件订阅器
    public class RecordBoilerInfo
    {
        static void Logger(string info)
        {
            Console.WriteLine(info);
        }//end of Logger

        static void Main(string[] args)
        {
            BoilerInfoLogger filelog = new BoilerInfoLogger("e:\\boiler.txt");
            DelegateBoilerEvent boilerEvent = new DelegateBoilerEvent();
            boilerEvent.BoilerEventLog += new
            DelegateBoilerEvent.BoilerLogHandler(Logger);
            boilerEvent.BoilerEventLog += new
            DelegateBoilerEvent.BoilerLogHandler(filelog.Logger);
            boilerEvent.LogProcess();
            Console.ReadLine();
            filelog.Close();
        }//end of main

    }//end of RecordBoilerInfo
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Logging info:

Temperature 100
Pressure 12

Message: O. K
```

C# 集合（Collection）

集合（Collection）类是专门用于数据存储和检索的类。这些类提供了对栈（stack）、队列（queue）、列表（list）和哈希表（hash table）的支持。大多数集合类实现了相同的接口。

集合（Collection）类服务于不同的目的，如为元素动态分配内存，基于索引访问列表项等等。这些类创建 Object 类的对象的集合。在 C# 中，Object 类是所有数据类型的基类。

各种集合类和它们的用法

下面是各种常用的 **System.Collection** 命名空间的类。点击下面的链接查看细节。

类	描述和用法
动态数组（ArrayList）	它代表了可被单独索引的对象的有序集合。 它基本上可以替代一个数组。但是，与数组不同的是，您可以使用索引在指定的位置添加和移除项目，动态数组会自动重新调整它的大小。它也允许在列表中进行动态内存分配、增加、搜索、排序各项。
哈希表（Hashtable）	它使用键来访问集合中的元素。 当您使用键访问元素时，则使用哈希表，而且您可以识别一个有用的键值。哈希表中的每一项都有一个键/值对。键用于访问集合中的项目。
排序列表（SortedList）	它可以使用键和索引来访问列表中的项。 排序列表是数组和哈希表的组合。它包含一个可使用键或索引访问各项的列表。如果您使用索引访问各项，则它是一个动态数组（ArrayList），如果您使用键访问各项，则它是一个哈希表（Hashtable）。集合中的各项总是按键值排序。
堆栈（Stack）	它代表了一个后进先出的对象集合。 当您需要对各项进行后进先出的访问时，则使用堆栈。当您在列表中添加一项，称为推入元素，当您从列表中移除一项时，称为弹出元素。
队列（Queue）	它代表了一个先进先出的对象集合。 当您需要对各项进行先进先出的访问时，则使用队列。当您在列表中添加一项，称为入队，当您从列表中移除一项时，称为出队。
点阵列（BitArray）	它代表了一个使用值 1 和 0 来表示的二进制数组。 当您需要存储位，但是事先不知道位数时，则使用点阵列。您可以使用整型索引从点阵列集合中访问各项，索引从零开始。

C# 泛型（Generic）

泛型（**Generic**） 允许您延迟编写类或方法中的编程元素的数据类型的规范，直到实际在程序中使用它的时候。换句话说，泛型允许您编写一个可以与任何数据类型一起工作的类或方法。

您可以通过数据类型的替代参数编写类或方法的规范。当编译器遇到类的构造函数或方法的函数调用时，它会生成代码来处理指定的数据类型。下面这个简单的实例将有助于您理解这个概念：

```
using System;
using System.Collections.Generic;

namespace GenericApplication
{
    public class MyGenericArray<T>
    {
        private T[] array;
        public MyGenericArray(int size)
        {
            array = new T[size + 1];
        }
        public T getItem(int index)
        {
            return array[index];
        }
        public void setItem(int index, T value)
        {
            array[index] = value;
        }
    }

    class Tester
    {
        static void Main(string[] args)
        {
            // 声明一个整型数组
            MyGenericArray<int> intArray = new MyGenericArray<int>(5);
            // 设置值
            for (int c = 0; c < 5; c++)
            {
                intArray.setItem(c, c*5);
            }
            // 获取值
            for (int c = 0; c < 5; c++)
            {
                Console.Write(intArray.getItem(c) + " ");
            }
            Console.WriteLine();
            // 声明一个字符数组
            MyGenericArray<char> charArray = new MyGenericArray<char>(5);
            // 设置值
            for (int c = 0; c < 5; c++)
            {
                charArray.setItem(c, (char)(c+97));
            }
            // 获取值
            for (int c = 0; c < 5; c++)
            {
                Console.Write(charArray.getItem(c) + " ");
            }
            Console.WriteLine();
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
0 5 10 15 20
a b c d e
```

泛型（Generic）的特性

使用泛型是一种增强程序功能的技术，具体表现在以下几个方面：

- 它有助于您最大限度地重用代码、保护类型的安全以及提高性能。
- 您可以创建泛型集合类。.NET 框架类库在 *System.Collections.Generic* 命名空间中包含了一些新的泛型集合类。您可以使用这些泛型集合类来替代 *System.Collections* 中的集合类。
- 您可以创建自己的泛型接口、泛型类、泛型方法、泛型事件和泛型委托。
- 您可以对泛型类进行约束以访问特定数据类型的方法。
- 关于泛型数据类型中使用的类型的信息可在运行时通过使用反射获取。

泛型（Generic）方法

在上面的实例中，我们已经使用了泛型类，我们可以通过类型参数声明泛型方法。下面的程序说明了这个概念：

```
using System;
using System.Collections.Generic;

namespace GenericMethodAppl
{
    class Program
    {
        static void Swap<T>(ref T lhs, ref T rhs)
        {
            T temp;
            temp = lhs;
            lhs = rhs;
            rhs = temp;
        }
        static void Main(string[] args)
        {
            int a, b;
            char c, d;
            a = 10;
            b = 20;
            c = 'I';
            d = 'V';

            // 在交换之前显示值
            Console.WriteLine("Int values before calling swap:");
            Console.WriteLine("a = {0}, b = {1}", a, b);
            Console.WriteLine("Char values before calling swap:");
            Console.WriteLine("c = {0}, d = {1}", c, d);

            // 调用 swap
            Swap<int>(ref a, ref b);
            Swap<char>(ref c, ref d);

            // 在交换之后显示值
            Console.WriteLine("Int values after calling swap:");
            Console.WriteLine("a = {0}, b = {1}", a, b);
            Console.WriteLine("Char values after calling swap:");
            Console.WriteLine("c = {0}, d = {1}", c, d);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Int values before calling swap:
a = 10, b = 20
Char values before calling swap:
c = I, d = V
Int values after calling swap:
a = 20, b = 10
Char values after calling swap:
c = V, d = I
```

泛型（Generic）委托

您可以通过类型参数定义泛型委托。例如：

```
delegate T NumberChanger<T>(T n);
```

下面的实例演示了委托的使用：

```
using System;
using System.Collections.Generic;

delegate T NumberChanger<T>(T n);
namespace GenericDelegateAppl
{
    class TestDelegate
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }

        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }

        static void Main(string[] args)
        {
            // 创建委托实例
            NumberChanger<int> nc1 = new NumberChanger<int>(AddNum);
            NumberChanger<int> nc2 = new NumberChanger<int>(MultNum);
            // 使用委托对象调用方法
            nc1(25);
            Console.WriteLine("Value of Num: {0}", getNum());
            nc2(5);
            Console.WriteLine("Value of Num: {0}", getNum());
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Value of Num: 35
Value of Num: 175
```

C# 匿名方法

我们已经提到过，委托是用于引用与其具有相同标签的方法。换句话说，您可以使用委托对象调用可由委托引用的方法。

匿名方法（**Anonymous methods**） 提供了一种传递代码块作为委托参数的技术。匿名方法是没有名称只有主体的方法。

在匿名方法中您不需要指定返回类型，它是从方法主体内的 return 语句推断的。

编写匿名方法的语法

匿名方法是通过使用 **delegate** 关键字创建委托实例来声明的。例如：

```
delegate void NumberChanger(int n);
...
NumberChanger nc = delegate(int x)
{
    Console.WriteLine("Anonymous Method: {0}", x);
};
```

代码块 *Console.WriteLine("Anonymous Method: {0}", x);* 是匿名方法的主体。

委托可以通过匿名方法调用，也可以通过命名方法调用，即，通过向委托对象传递方法参数。

例如：

```
nc(10);
```

实例

下面的实例演示了匿名方法的概念：

```
using System;

delegate void NumberChanger(int n);
namespace DelegateAppl
{
    class TestDelegate
    {
        static int num = 10;
        public static void AddNum(int p)
        {
            num += p;
            Console.WriteLine("Named Method: {0}", num);
        }

        public static void MultNum(int q)
        {
            num *= q;
            Console.WriteLine("Named Method: {0}", num);
        }
        public static int getNum()
        {
            return num;
        }

        static void Main(string[] args)
        {
            // 使用匿名方法创建委托实例
            NumberChanger nc = delegate(int x)
            {
               Console.WriteLine("Anonymous Method: {0}", x);
            };

            // 使用匿名方法调用委托
            nc(10);

            // 使用命名方法实例化委托
            nc =  new NumberChanger(AddNum);

            // 使用命名方法调用委托
            nc(5);

            // 使用另一个命名方法实例化委托
            nc =  new NumberChanger(MultNum);

            // 使用命名方法调用委托
            nc(2);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Anonymous Method: 10
Named Method: 15
Named Method: 30
```

C# 不安全代码

当一个代码块使用 **unsafe** 修饰符标记时，C# 允许在函数中使用指针变量。不安全代码或非托管代码是指使用了指针变量的代码块。

指针变量

指针 是值为另一个变量的地址的变量，即，内存位置的直接地址。就像其他变量或常量，您必须在使用指针存储其他变量地址之前声明指针。

指针变量声明的一般形式为：

```
type *var-name;
```

以下是有效的指针声明：

```
int    *ip;    /* 指向一个整数 */
double *dp;    /* 指向一个双精度数 */
float  *fp;    /* 指向一个浮点数 */
char   *ch     /* 指向一个字符 */
```

下面的实例说明了 C# 中使用了 **unsafe** 修饰符时指针的使用：

```
using System;
namespace UnsafeCodeApplication
{
    class Program
    {
        static unsafe void Main(string[] args)
        {
            int var = 20;
            int* p = &var;
            Console.WriteLine("Data is: {0} ",  var);
            Console.WriteLine("Address is: {0}",  (int)p);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Data is: 20
Address is: 99215364
```

您也可以不用声明整个方法作为不安全代码，只需要声明方法的一部分作为不安全代码。下面的实例说明了这点。

使用指针检索数据值

您可以使用 **ToString()** 方法检索存储在指针变量所引用位置的数据。下面的实例演示了这点：

```
using System;
namespace UnsafeCodeApplication
{
   class Program
   {
      public static void Main()
      {
         unsafe
         {
            int var = 20;
            int* p = &var;
            Console.WriteLine("Data is: {0} " , var);
            Console.WriteLine("Data is: {0} " , p->ToString());
            Console.WriteLine("Address is: {0} " , (int)p);
         }
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Data is: 20
Data is: 20
Address is: 77128984
```

传递指针作为方法的参数

您可以向方法传递指针变量作为方法的参数。下面的实例说明了这点：

```
using System;
namespace UnsafeCodeApplication
{
   class TestPointer
   {
      public unsafe void swap(int* p, int *q)
      {
         int temp = *p;
         *p = *q;
         *q = temp;
      }

      public unsafe static void Main()
      {
         TestPointer p = new TestPointer();
         int var1 = 10;
         int var2 = 20;
         int* x = &var1;
         int* y = &var2;

         Console.WriteLine("Before Swap: var1:{0}, var2: {1}", var1, var2);
         p.swap(x, y);

         Console.WriteLine("After Swap: var1:{0}, var2: {1}", var1, var2);
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Before Swap: var1: 10, var2: 20
After Swap: var1: 20, var2: 10
```

使用指针访问数组元素

在 C# 中，数组名称和一个指向与数组数据具有相同数据类型的指针是不同的变量类型。例如，int *p 和 int[] p 是不同的类型。您可以增加指针变量 p，因为它在内存中不是固定的，但是数组地址在内存中是固定的，所以您不能增加数组 p。

因此，如果您需要使用指针变量访问数组数据，可以像我们通常在 C 或 C++ 中所做的那样，使用 **fixed** 关键字来固定指针。

下面的实例演示了这点：

```
using System;
namespace UnsafeCodeApplication
{
   class TestPointer
   {
      public unsafe static void Main()
      {
         int[]  list = {10, 100, 200};
         fixed(int *ptr = list)

         /* 显示指针中数组地址 */
         for ( int i = 0; i < 3; i++)
         {
            Console.WriteLine("Address of list[{0}]={1}",i,(int)(ptr + i));
            Console.WriteLine("Value of list[{0}]={1}", i, *(ptr + i));
         }
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Address of list[0] = 31627168
Value of list[0] = 10
Address of list[1] = 31627172
Value of list[1] = 100
Address of list[2] = 31627176
Value of list[2] = 200
```

编译不安全代码

为了编译不安全代码，您必须切换到命令行编译器指定 **/unsafe** 命令行。

例如，为了编译包含不安全代码的名为 prog1.cs 的程序，需在命令行中输入命令：

```
csc /unsafe prog1.cs
```

如果您使用的是 Visual Studio IDE，那么您需要在项目属性中启用不安全代码。

步骤如下：

- 通过双击资源管理器（Solution Explorer）中的属性（properties）节点，打开项目属性（**project properties**）。
- 点击 **Build** 标签页。
- 选择选项"**Allow unsafe code**"。

C# 多线程

线程 被定义为程序的执行路径。每个线程都定义了一个独特的控制流。如果您的应用程序涉及到复杂的和耗时的操作，那么设置不同的线程执行路径往往是有益的，每个线程执行特定的工作。

线程是轻量级进程。一个使用线程的常见实例是现代操作系统中并行编程的实现。使用线程节省了 CPU 周期的浪费，同时提高了应用程序的效率。

到目前为止我们编写的程序是一个单线程作为应用程序的运行实例的单一的过程运行的。但是，这样子应用程序同时只能执行一个任务。为了同时执行多个任务，它可以被划分为更小的线程。

线程生命周期

线程生命周期开始于 System.Threading.Thread 类的对象被创建时，结束于线程被终止或完成执行时。

下面列出了线程生命周期中的各种状态：

- 未启动状态：当线程实例被创建但 Start 方法未被调用时的状况。
- 就绪状态：当线程准备好运行并等待 CPU 周期时的状况。
- 不可运行状态：下面的几种情况下线程是不可运行的：
  - 已经调用 Sleep 方法
  - 已经调用 Wait 方法
  - 通过 I/O 操作阻塞
- 死亡状态：当线程已完成执行或已中止时的状况。

主线程

在 C# 中，**System.Threading.Thread** 类用于线程的工作。它允许创建并访问多线程应用程序中的单个线程。进程中第一个被执行的线程称为主线程。

当 C# 程序开始执行时，主线程自动创建。使用 **Thread** 类创建的线程被主线程的子线程调用。您可以使用 Thread 类的 **CurrentThread** 属性访问线程。

下面的程序演示了主线程的执行：

```
using System;
using System.Threading;

namespace MultithreadingApplication
{
    class MainThreadProgram
    {
        static void Main(string[] args)
        {
            Thread th = Thread.CurrentThread;
            th.Name = "MainThread";
            Console.WriteLine("This is {0}", th.Name);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
This is MainThread
```

Thread 类常用的属性和方法

下表列出了 **Thread** 类的一些常用的 属性：

属性	描述
CurrentContext	获取线程正在其中执行的当前上下文。
CurrentCulture	获取或设置当前线程的区域性。
CurrentPrinciple	获取或设置线程的当前负责人（对基于角色的安全性而言）。
CurrentThread	获取当前正在运行的线程。
CurrentUICulture	获取或设置资源管理器使用的当前区域性以便在运行时查找区域性特定的资源。
ExecutionContext	获取一个 ExecutionContext 对象，该对象包含有关当前线程的各种上下文的信息。
IsAlive	获取一个值，该值指示当前线程的执行状态。
IsBackground	获取或设置一个值，该值指示某个线程是否为后台线程。
IsThreadPoolThread	获取一个值，该值指示线程是否属于托管线程池。
ManagedThreadId	获取当前托管线程的唯一标识符。
Name	获取或设置线程的名称。
Priority	获取或设置一个值，该值指示线程的调度优先级。
ThreadState	获取一个值，该值包含当前线程的状态。

下表列出了 **Thread** 类的一些常用的 方法：

方法名	描述
public void Abort()	在调用此方法的线程上引发 ThreadAbortException，以开始终止此线程的过程。调用此方法通常会终止线程。
public static LocalDataStoreSlot AllocateDataSlot()	在所有的线程上分配未命名的数据槽。为了获得更好的性能，请改用以 ThreadStaticAttribute 属性标记的字段。
public static LocalDataStoreSlot AllocateNamedDataSlot(string name)	在所有线程上分配已命名的数据槽。为了获得更好的性能，请改用以 ThreadStaticAttribute 属性标记的字段。
public static void BeginCriticalRegion()	通知主机执行将要进入一个代码区域，在该代码区域内线程中止或未经处理的异常的影响可能会危害应用程序域中的其他任务。
public static void BeginThreadAffinity()	通知主机托管代码将要执行依赖于当前物理操作系统线程的标识的指令。
public static void EndCriticalRegion()	通知主机执行将要进入一个代码区域，在该代码区域内线程中止或未经处理的异常仅影响当前任务。
public static void EndThreadAffinity()	通知主机托管代码已执行完依赖于当前物理操作系统线程的标识的指令。
public static void FreeNamedDataSlot(string name)	为进程中的所有线程消除名称与槽之间的关联。为了获得更好的性能，请改用以 ThreadStaticAttribute 属性标记的字段。
public static Object GetData(LocalDataStoreSlot slot)	在当前线程的当前域中从当前线程上指定的槽中检索值。为了获得更好的性能，请改用以 ThreadStaticAttribute 属性标记的字段。
public static AppDomain GetDomain()	返回当前线程正在其中运行的当前域。
public static AppDomain GetDomainID()	返回唯一的应用程序域标识符。
public static LocalDataStoreSlot GetNamedDataSlot(string name)	查找已命名的数据槽。为了获得更好的性能，请改用以 ThreadStaticAttribute 属性标记的字段。

public void Interrupt()	中断处于 WaitSleepJoin 线程状态的线程。
public void Join()	在继续执行标准的 COM 和 SendMessage 消息泵处理期间，阻塞调用线程，直到某个线程终止为止。此方法有不同的重载形式。
public static void MemoryBarrier()	按如下方式同步内存存取：执行当前线程的处理器在对指令重新排序时，不能采用先执行 MemoryBarrier 调用之后的内存存取，再执行 MemoryBarrier 调用之前的内存存取的方式。
public static void ResetAbort()	取消为当前线程请求的 Abort。
public static void SetData(LocalDataStoreSlot slot, Object data)	在当前正在运行的线程上为此线程的当前域在指定槽中设置数据。为了获得更好的性能，请改用以 ThreadStaticAttribute 属性标记的字段。
public void Start()	开始一个线程。
public static void Sleep(int millisecondsTimeout)	让线程暂停一段时间。
public static void SpinWait(int iterations)	导致线程等待由 iterations 参数定义的时间量。
public static byte VolatileRead(ref byte address) public static double VolatileRead(ref double address) public static int VolatileRead(ref int address) public static Object VolatileRead(ref Object address)	读取字段值。无论处理器的数目或处理器缓存的状态如何，该值都是由计算机的任何处理器写入的最新值。此方法有不同的重载形式。这里只给出了一些形式。
public static void VolatileWrite(ref byte address, byte value) public static void VolatileWrite(ref double address, double value) public static void VolatileWrite(ref int address, int value) public static void VolatileWrite(ref Object address, Object value)	立即向字段写入一个值，以使该值对计算机中的所有处理器都可见。此方法有不同的重载形式。这里只给出了一些形式。
public static bool Yield()	导致调用线程执行准备好在当前处理器上运行的另一个线程。由操作系统选择要执行的线程。

创建线程

线程是通过扩展 Thread 类创建的。扩展的 Thread 类调用 **Start()** 方法来开始子线程的执行。

下面的程序演示了这个概念：

```
using System;
using System.Threading;

namespace MultithreadingApplication
{
    class ThreadCreationProgram
    {
        public static void CallToChildThread()
        {
            Console.WriteLine("Child thread starts");
        }

        static void Main(string[] args)
        {
            ThreadStart childref = new ThreadStart(CallToChildThread);
            Console.WriteLine("In Main: Creating the Child thread");
            Thread childThread = new Thread(childref);
            childThread.Start();
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
In Main: Creating the Child thread
Child thread starts
```

管理线程

Thread 类提供了各种管理线程的方法。

下面的实例演示了 **sleep()** 方法的使用，用于在一个特定的时间暂停线程。

```
using System;
using System.Threading;

namespace MultithreadingApplication
{
    class ThreadCreationProgram
    {
        public static void CallToChildThread()
        {
            Console.WriteLine("Child thread starts");
            // 线程暂停 5000 毫秒
            int sleepfor = 5000;
            Console.WriteLine("Child Thread Paused for {0} seconds",
                              sleepfor / 1000);
            Thread.Sleep(sleepfor);
            Console.WriteLine("Child thread resumes");
        }

        static void Main(string[] args)
        {
            ThreadStart childref = new ThreadStart(CallToChildThread);
            Console.WriteLine("In Main: Creating the Child thread");
            Thread childThread = new Thread(childref);
            childThread.Start();
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
In Main: Creating the Child thread
Child thread starts
Child Thread Paused for 5 seconds
Child thread resumes
```

销毁线程

Abort() 方法用于销毁线程。

通过抛出 **threadabortexception** 在运行时中止线程。这个异常不能被捕获，如果有 *finally* 块，控制会被送至 *finally* 块。

下面的程序说明了这点：

```
using System;
using System.Threading;

namespace MultithreadingApplication
{
    class ThreadCreationProgram
    {
        public static void CallToChildThread()
        {
            try
            {

                Console.WriteLine("Child thread starts");
                // 计数到 10
                for (int counter = 0; counter <= 10; counter++)
                {
                    Thread.Sleep(500);
                    Console.WriteLine(counter);
                }
                Console.WriteLine("Child Thread Completed");

            }
            catch (ThreadAbortException e)
            {
                Console.WriteLine("Thread Abort Exception");
            }
            finally
            {
                Console.WriteLine("Couldn't catch the Thread Exception");
            }

        }

        static void Main(string[] args)
        {
            ThreadStart childref = new ThreadStart(CallToChildThread);
            Console.WriteLine("In Main: Creating the Child thread");
            Thread childThread = new Thread(childref);
            childThread.Start();
            // 停止主线程一段时间
            Thread.Sleep(2000);
            // 现在中止子线程
            Console.WriteLine("In Main: Aborting the Child thread");
            childThread.Abort();
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
In Main: Creating the Child thread
Child thread starts
0
1
2
In Main: Aborting the Child thread
Thread Abort Exception
Couldn't catch the Thread Exception
```

Excel教程

Excel是微软办公套装软件(Microsoft office的组件之一)的一个重要的组成部分，它可以进行各种数据的处理、统计分析和辅助决策操作，广泛地应用于管理、统计财经、金融等众多领域。

Microsoft Excel是微软Microsoft编写并分发Windows和Mac OS X商用电子表格应用程序，在写这篇教程的时候的当前版本是微软Windows Excel2010和Mac OS X Excel 2011。

Microsoft Excel是一个电子表格工具，能够进行计算，分析数据和整合来自不同程序的信息。默认情况下，保存在Excel2010文档保存为扩展名为.xlsx，Excel之前版本的文件扩展名是.xls。

本章将教你如何在简单的步骤，启动Excel 2010应用程序。 假设您已经将Microsoft Office2010安装在你的电脑上，要启动Excel应用程序，请在您的电脑以下步骤：

步骤 (1): 点击 开始 按钮。



步骤 **(2):** 从菜单中单击 所有程序 选项。



All Programs

步骤 **(3):** 从子菜单搜索***Microsoft Office***，然后单击它。



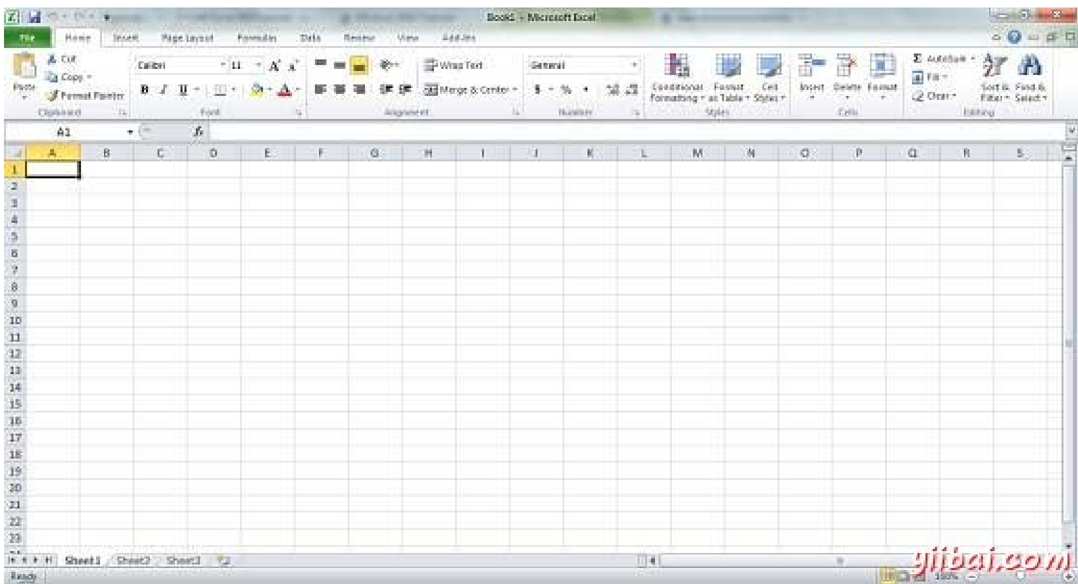
Microsoft Office

步骤 **(4):** 从子菜单中搜索***Microsoft Excel2010***，点击它。



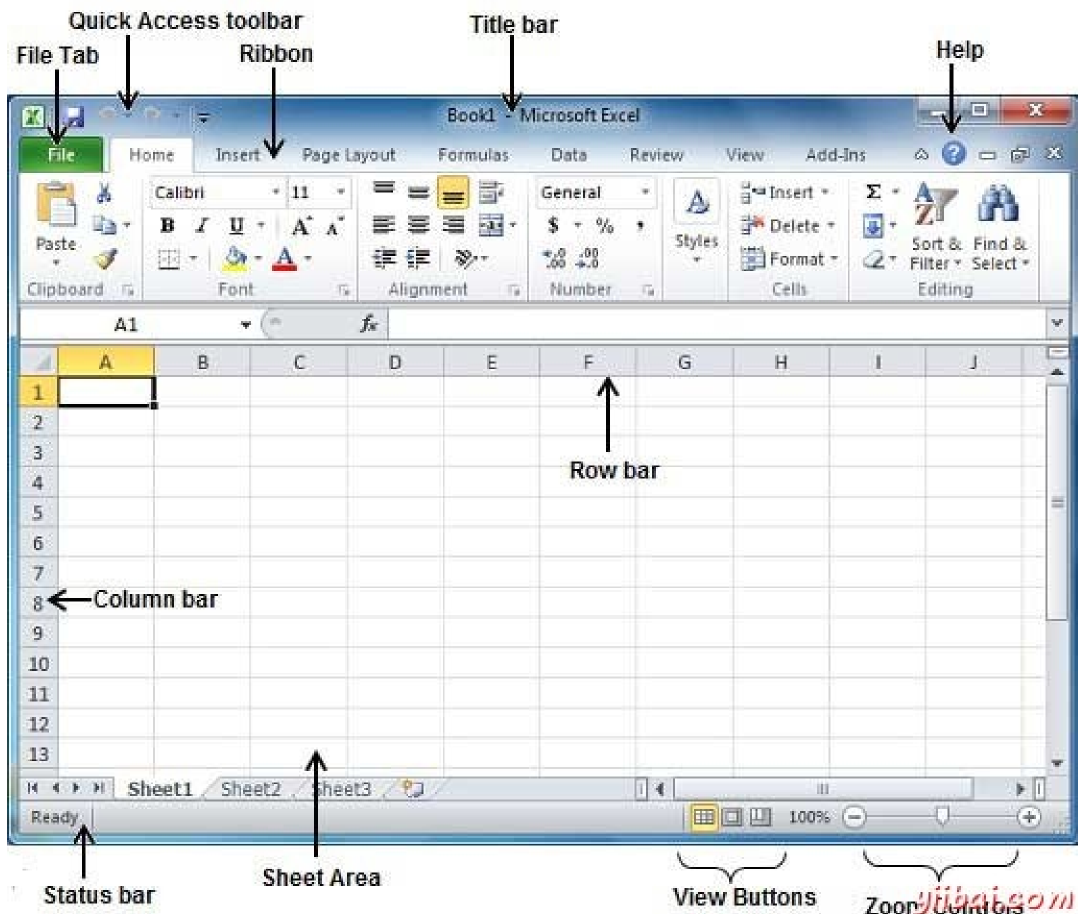
Microsoft Excel 2010

这将启动 **Microsoft Excel 2010** 应用程序，会看到下面的Excel窗口。



Excel浏览窗口 - Excel教程

当启动Excel应用程序，会显示基本的窗口(界面)。让我们了解这个窗口的各个重要组成部分。



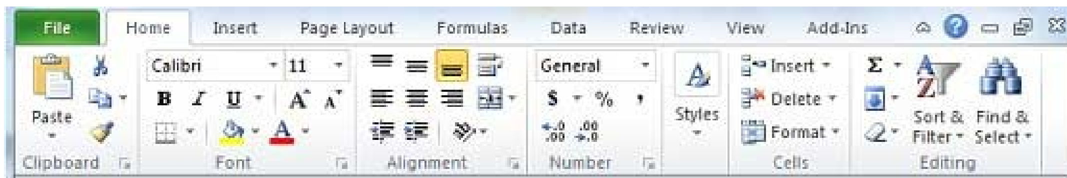
文件选项卡：

该文件选项卡取代Excel 2007中的"办公"按钮，您可以点击它来查看Backstage视图， 这是当你需要打开或保存文件，创建新表，打印纸，并做其他文件相关操作的地方。

快速访问工具栏：

这里你会发现正上方的File选项卡，其目的是提供一个方便的地方以Excel最常用的命令。您可以自定义这个工具栏根据您的要求。

功能区：



功能区包含以下三个组成部分的命令：

- 选项卡：它们显示在整个功能区的顶部，并包含相关命令的群体。主页，插入，页面布局的例子在功能区选项卡中。
- 分组: 他们组织相关的命令;每个组的名字出现在组下方的功能区。对于有关字体或一组命令相关对准指令等例子组
- 命令: 命令显示每个分组内如上所述。

标题栏：

这个位于中间和顶部或窗口。标题栏显示程序和表的标题。

帮助：

帮助图标可以被用来随时获取相关excel 有关帮助。这提供了Excel相关的各学科不错的教程。

缩放控制：

缩放控制可让您放大仔细看看文字。缩放控制包含向左或向右滑动来放大或缩小一个滑块，-和+按钮，可以点击用来增加或减少缩放系数。

视图按钮：

本组有三个按钮位于缩放控制的左侧，在屏幕的底部附近，可以让Excel在各种表视图之间切换。

- 普通版式视图：这在普通视图中显示页。
- 页面布局视图 :这将显示页面在打印时出现完全一样。这给出了一个全屏幕的外观的文档。
- 分页符视图：这显示在打印时，其中的页面将换页预览。

Sheet区域：

输入数据的区域。闪烁的竖条称为插入点，它代表在那里将文本键入时显示的位置。

行栏

行编号从1开始，并在不断输入数据时增加。最多可输入1,048,576行。

列栏

列编号从A开始并保持对输入数据增加。在Z之后，将开始一系列的AA，AB等。最高大小为16,384列。

状态栏：

这将显示表的信息以及所述插入点位置。从左至右，此栏可以包含的文档中的页面和词语，语言等的总数

通过右键单击状态栏，可以在任何地方通过选择或不选择从提供的列表选项来配置。

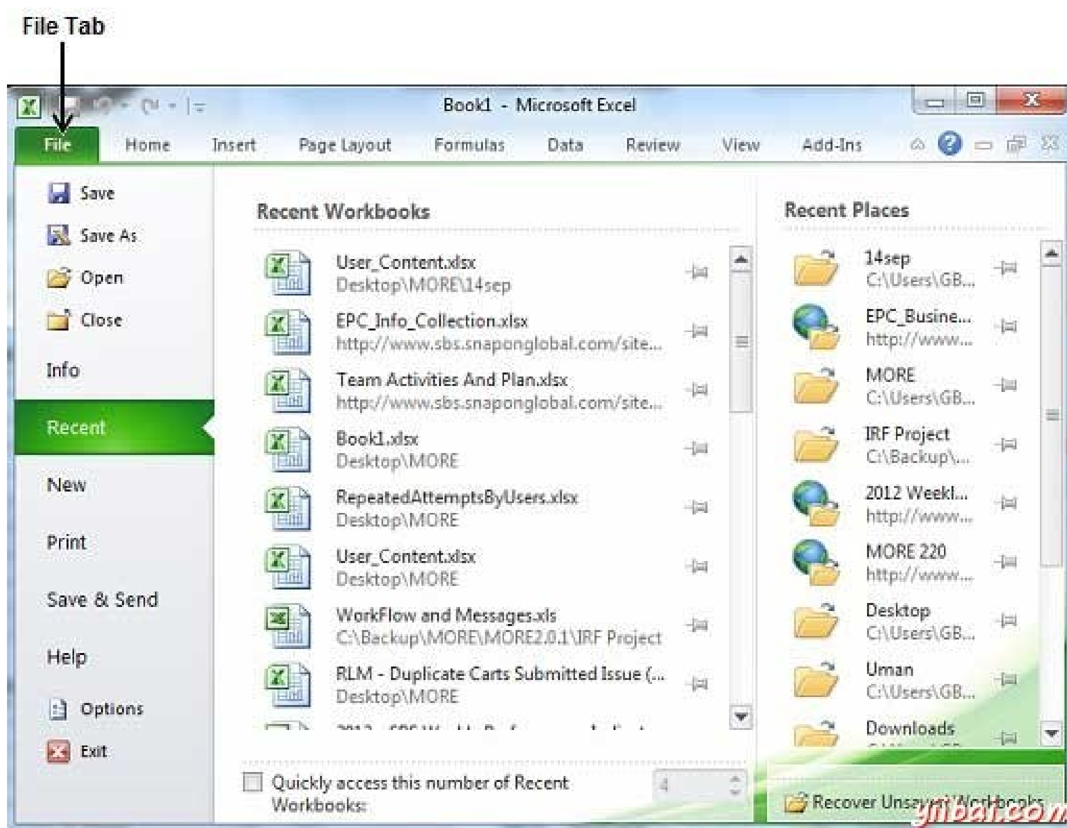
对话框启动器：

这是在多组功能区的右下角非常小箭头。单击此按钮会打开一个对话框或任务窗格中，提供有关组更多的选择。

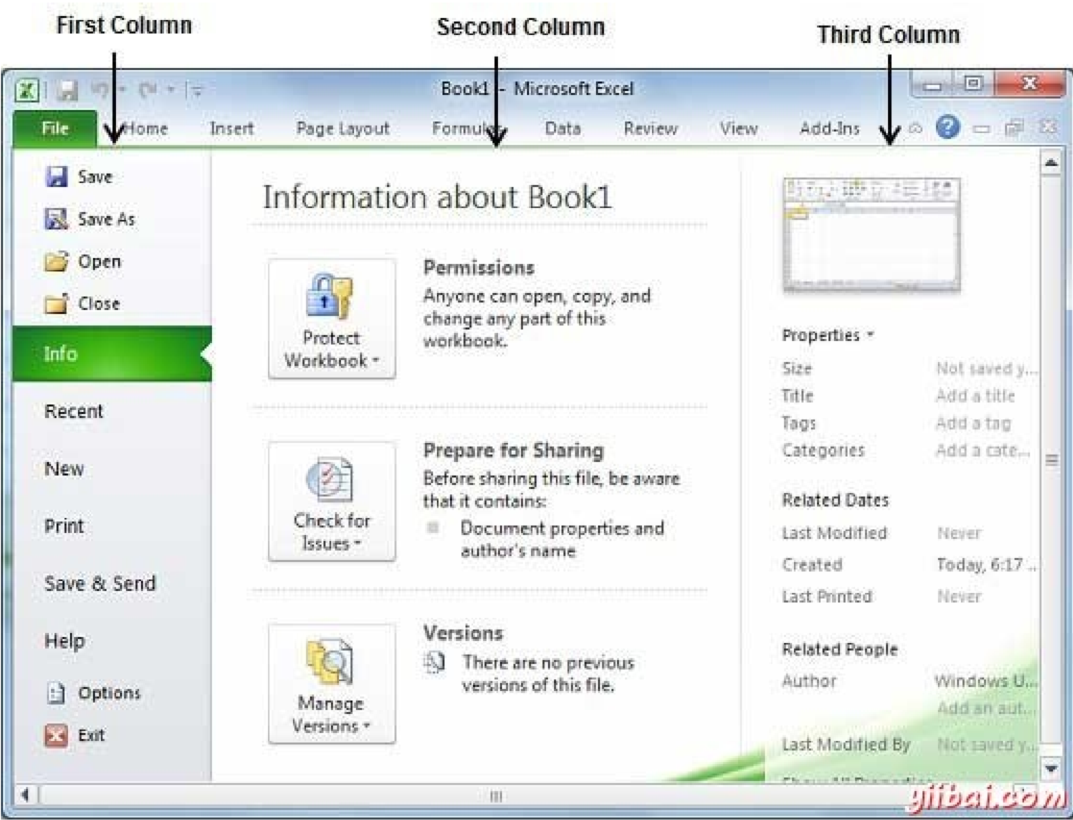
Excel BackStage视图 - Excel教程

Backstage视图已在Excel 2010中被引入，并作为管理表的中心位置。Backstage视图有助于创建新的工作表，保存和打开表，打印和共享表等等。

进入Backstage视图非常简单：只需点击文件选项卡，位于Excel功能区的左上角。如果还没有打开任何表，那么将看到一个窗口，其中列出了所有最近打开的表如下：



如果已经有一个打开的表，那么它会显示一个窗口，显示打开表的细节，如下图所示。Backstage视图显示了三列，大部分会选择在第一列的可用选项。



Backstage视图的第一列将有以下选择：

选项	描述
Save	如果一个现有工作表被打开，它会被保存，否则它会显示一个对话框，询问工作表名称
Save As	一个对话框将显示询问表名和表类型，默认情况下，它会保存为工作表的2010格式的扩展 .xlsx
Open	此选项将用于打开现有的Excel工作表
Close	这个选项将被用来关闭一个打开的工作表
Info	此选项将显示已打开的表的信息
Recent	这个选项会列出所有的最近打开表
New	此选项将被用来打开一个新的工作表
Print	这个选项将被用于打印的打开工作表
Save & Send	此选项将保存打开的表，将显示选项，使用电子邮件等方式发送工作表
Help	您可以使用此选项来获得所需有关Excel2010的帮助
Options	使用此选项可以设置有关Excel 2010各种选项
Exit	使用此选项可以关闭该表并退出

工作表信息

在第一列可单击信息选项，它显示在Backstage视图的第二列的信息如下：

- 兼容模式: 如果该表不是一个原始Excel2007/2010板，转换按钮会出现在这里，使您能够轻松地更新它的格式。否则，这个类别不会出现。
- 权限: 可以使用此选项来保护您的Excel工作表。可以设置一个密码，这样没有人可以打开你的工作表，或者也可以锁定工作表，这样没有人可以编辑您的工作表。
- 准备共享: 本节主要介绍你应该知道工作表的重要信息，然后将其发送给其他人，比如开发的工作表中编辑的记录。
- 版本: 如果工作表已保存了几次，你能够从这里访问其以前的版本。

工作表属性

可在第一列单击信息选项，在Backstage视图第三列显示各种属性。这些属性包括纸张尺寸，标题，标签，分类等。

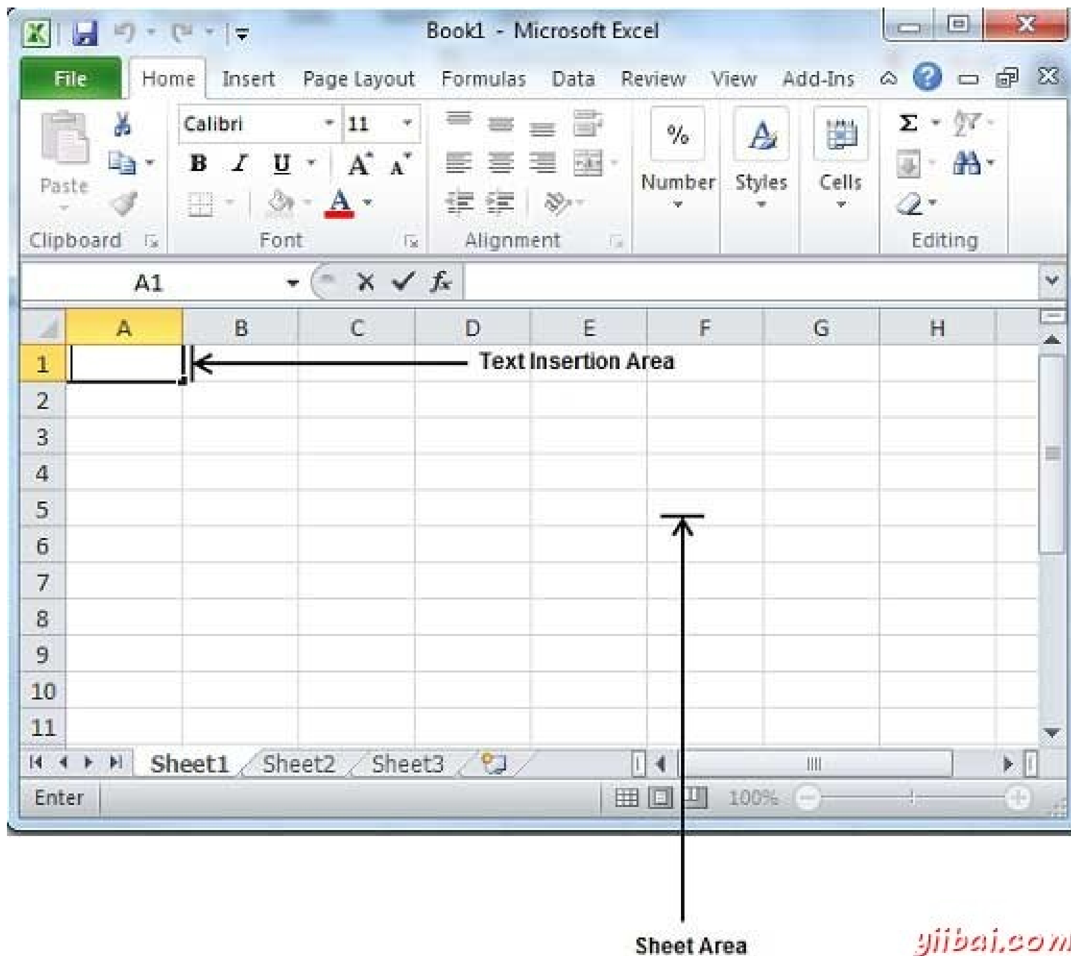
您也可以编辑各种属性。只是尝试单击属性值，如果属性是可编辑的，然后它会显示一个文本框，可以将您喜欢的文字标题，标签，注释，作者。

退出**Backstage**视图

要从Backstage视图中退出，只点击文件选项卡或在键盘上按Esc键，返回到Excel工作模式。

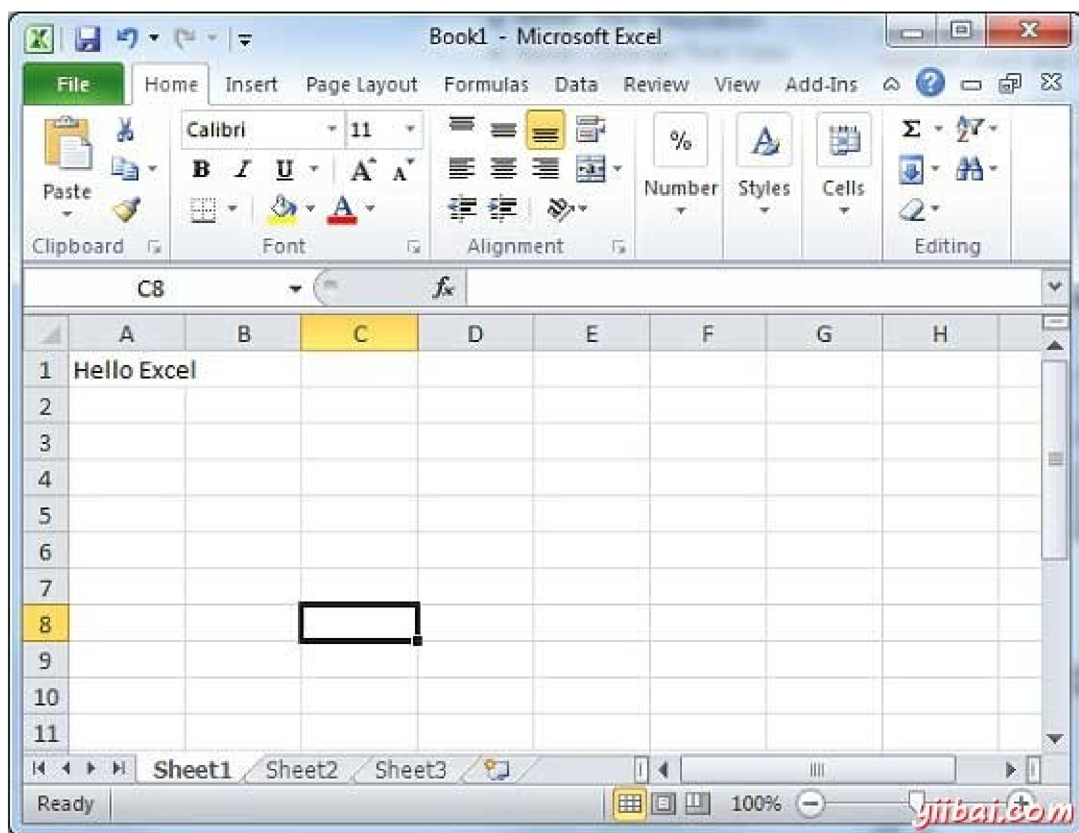
Excel中输入值 - Excel教程

让我们看看在Excel工作表中输入文本是多么容易。希望你了解，当启动一个工作表，它会显示一个新的工作表，在默认情况下，如下图所示：



工作表区域是键入文本的区域。闪烁的竖条称为插入点，它代表在那里将文本键入时显示的位置。 当你点击一个框，然后这个框会突出显示。当您双击框闪烁的竖条会来，此时输入数据即可。

因此，只要让鼠标光标在文本插入点，并键入您希望任何文本文字。我输入只有两个单词“**Hello Excel**”，如下图所示。键入的文本出现在插入点的左侧：



以下这三个要点将有助于你在打字时：

- 按Tab键转到下一列。
- 按Enter键进入下一行。
- 按下Alt + Enter键进入在同一列的新行。

Excel左右移动 - Excel教程

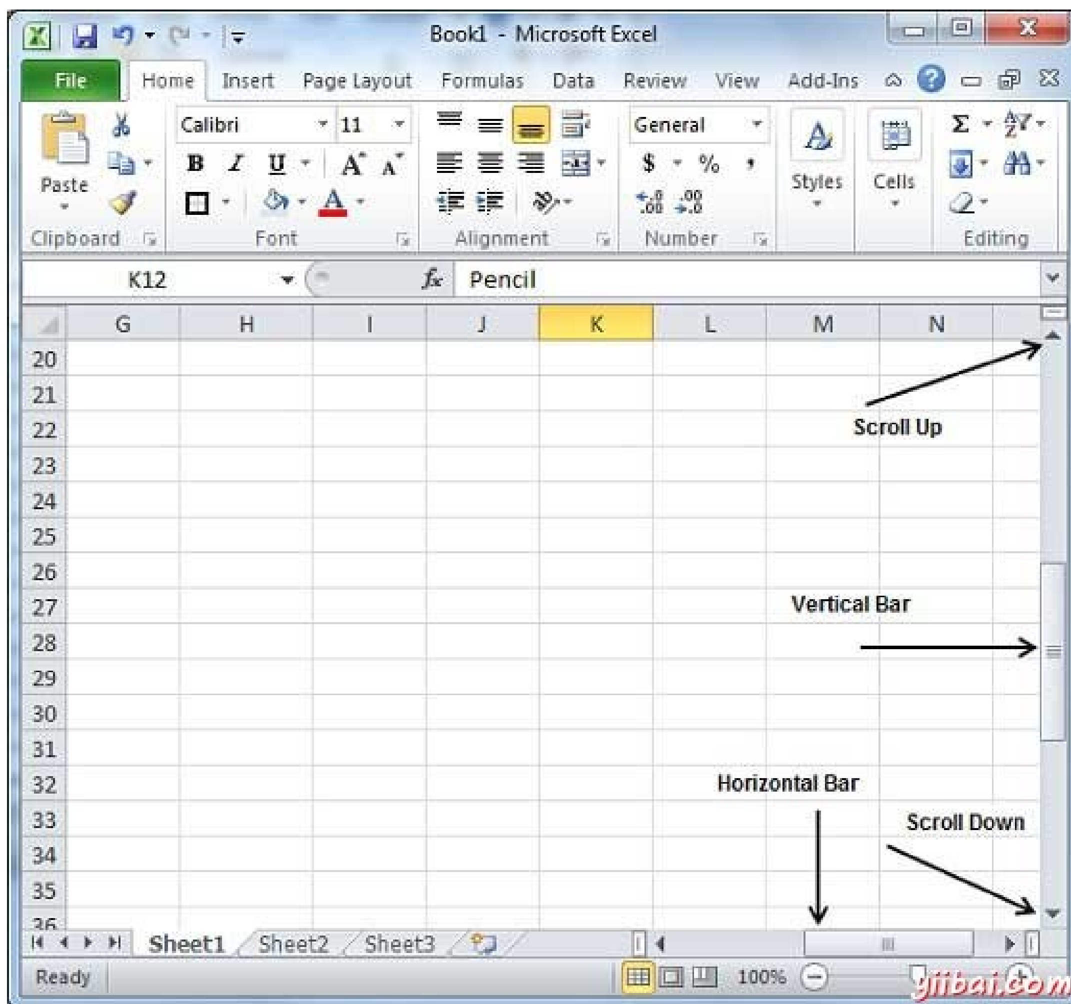
Excel提供多种方式移动，使用鼠标和键盘在工作表。

我们继续之前，首先让我们来创建一些示例文本。 打开一个新的Excel表，然后键入数据。我们已经截图所示的样本数据。

OrderDate	Region	Rep	Item	Units	Unit Cost	Total
1/6/2010	East	Jones	Pencil	95	1.99	189.05
1/23/2010	Central	Kivell	Binder	50	19.99	999.5
2/9/2010	Central	Jardine	Pencil	36	4.99	179.64
2/26/2010	Central	Gill	Pen	27	19.99	539.73
3/15/2010	West	Sorvino	Pencil	56	2.99	167.44
4/1/2010	East	Jones	Binder	60	4.99	299.4
4/18/2010	Central	Andrews	Pencil	75	1.99	149.25
5/5/2010	Central	Jardine	Pencil	90	4.99	449.1
5/22/2010	West	Thompson	Pencil	32	1.99	63.68
6/8/2010	East	Jones	Binder	60	8.99	539.4
6/25/2010	Central	Morgan	Pencil	90	4.99	449.1
7/12/2010	East	Howard	Binder	29	1.99	57.71
7/29/2010	East	Parent	Binder	81	19.99	1,619.19
8/15/2010	East	Jones	Pencil	35	4.99	174.65

使用鼠标移动

您可以通过点击任何地方的文字在屏幕上轻松地移动到插入点。有时候，如果表是很大，那么看不到你要移动的地方。在这种情况下，将不得不使用滚动条，如下面的屏幕截图：



可以通过滚动鼠标滚轮，这相当于点击滚动条的向上箭头或向下箭头按钮来滚动工作表。

使用移动滚动条

如图上面的屏幕捕获，有两个滚动条：一个用于片内垂直移动，和一个用于水平移动。使用垂直滚动条，您可以：

- 通过点击朝上的滚动箭头向上移动一行。
- 通过点击朝下的滚动箭头向下移动一行。
- 移动一个翻页，使用翻页键（脚注）。
- 移动一前一页面，使用前一页按钮（脚注）。
- 使用浏览对象（Browse Objec）按钮移动通过工作表，从一个选择的对象到下一个。

使用键盘移动

下面的键盘命令，用于您的工作表四处移动，也可移动到插入点：

Keystroke	其中，将移到到插入点
→	向前移动一个框
←	倒退一个框
↑	向上移动到一个框中
↓	向下移动到一个框中
PageUp	到上一个屏幕
PageDown	到下一个屏幕
Home	到当前屏幕的开始
End	到当前屏幕的结束

您可以移动框内或逐页。现在点击包含在工作表的任何框数据。需要按住Ctrl键的同时按箭头键，这将插入点移动，如下所述：

组合键	其中，将移到到插入点
Ctrl + →	包含当前行的数据到最后一个框中
Ctrl + ←	包含当前行的数据到第一框中
Ctrl + ↑	包含当前列的数据到第一框中
Ctrl + ↓	包含当前列的数据到最后一个框中
Ctrl + PageUp	在当前工作表到左边的工作表。
Ctrl + PageDown	在当前工作表到右边的工作表。
Ctrl + Home	到工作表的开始
Ctrl + End	到工作表的结束

使用定位命令移动

按F5键使用转到（Go To）命令，它会显示一个对话框，在这里将有多种选择来达到特定的盒子。

通常我们使用的行和列数，例如K5，按GO按钮。

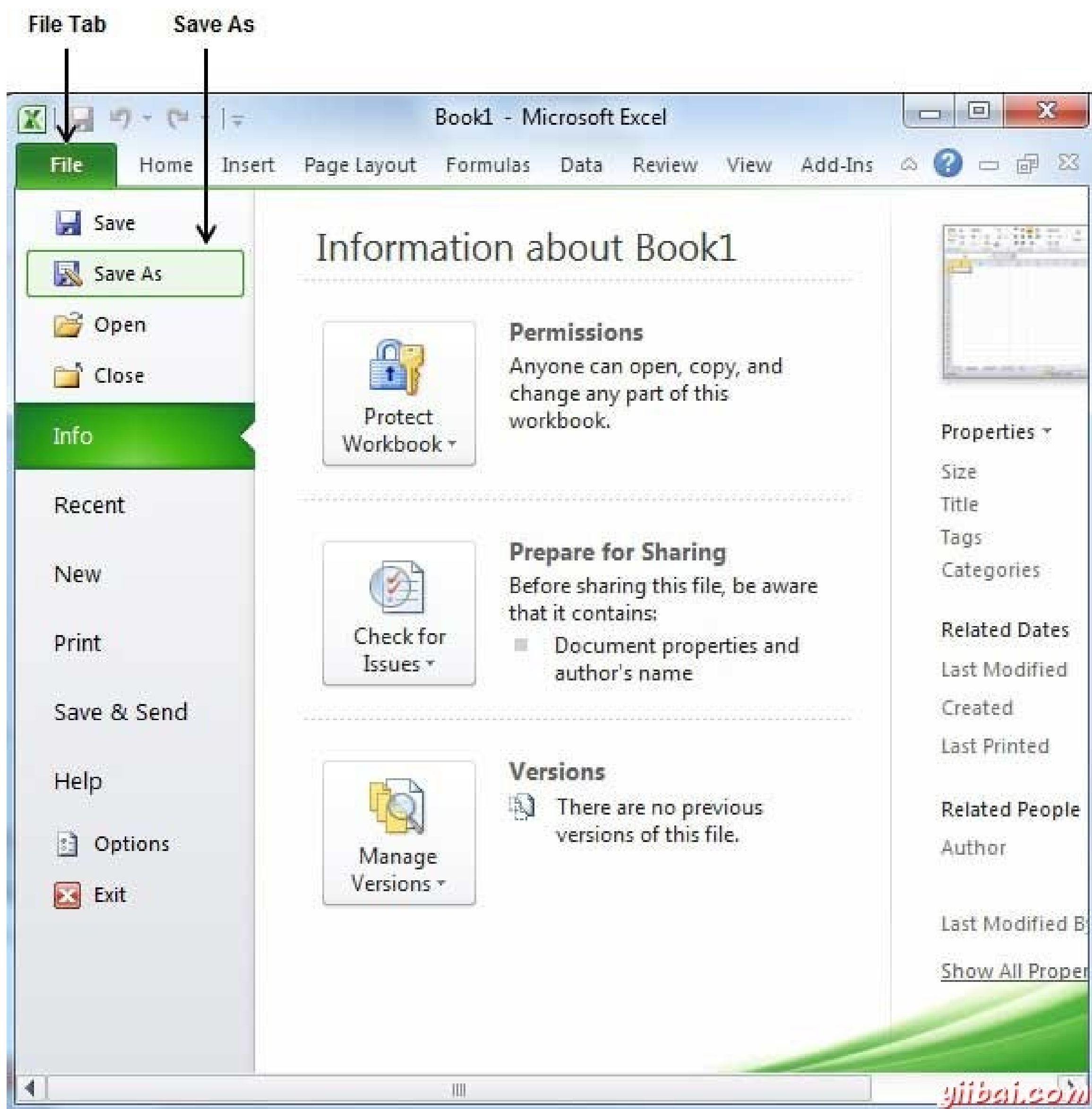

Go To
Go to:
K18
K25
O49
Reference:
k18
Special...
OK
Cancel
yiibai.com

Excel保存工作簿 - Excel教程

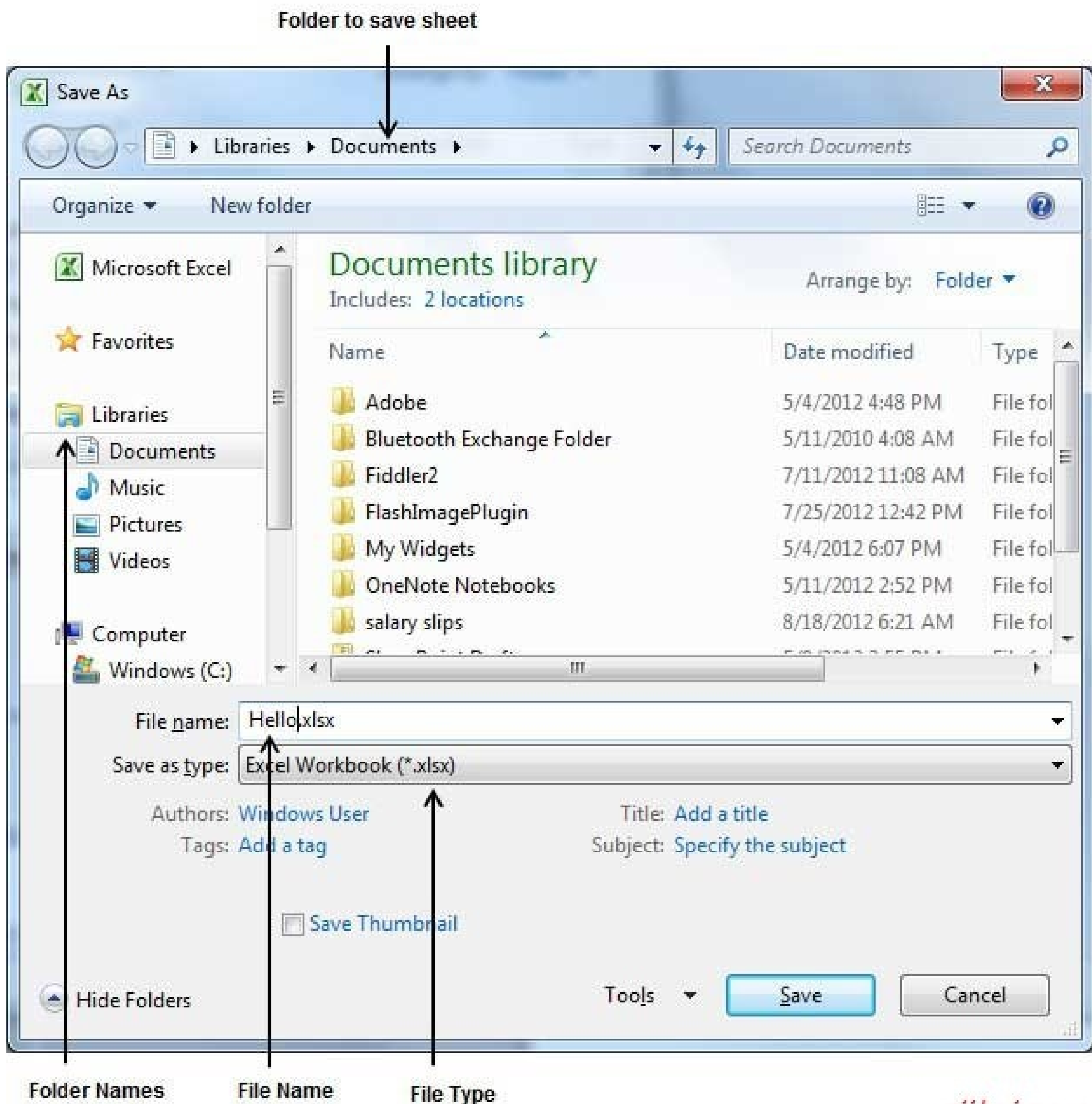
保存新工作表

当键入新的Excel工作表数据完成，是时候来保存您的工作表/工作簿，以避免丢失已在Excel表单所做的工作。 下面是保存编辑Excel表的步骤：

步骤(1)：点击文件选项卡(**File tab**)，选择另存为(**Save As**)选项。



步骤**(2)**：选择您想保存工作所在的文件夹，请输入想给工作表的文件名称，并选择保存类型的文件名，默认情况下是 **.xlsx** 格式。



步骤（3）：最后，点击保存按钮，您的工作表将被保存在所选文件夹中并使用输入的名称。

保存新变化

可能有一种情况，当打开一个现有的工作表，部分或完全编辑，甚至你想保存在工作表编辑的变化。如果想保存这个工作表使用相同的名称，那么可以使用下面简单的选择：

- 只要按下Ctrl+ S键保存更改。
- 也可以选择点击可用软盘图标左上角，在上面的文件选项卡。此选项也将保存更改。
- 还可以使用第三种方法来保存更改，这是可以一样显示在上面的屏幕截图另存为选项上面保存选项。

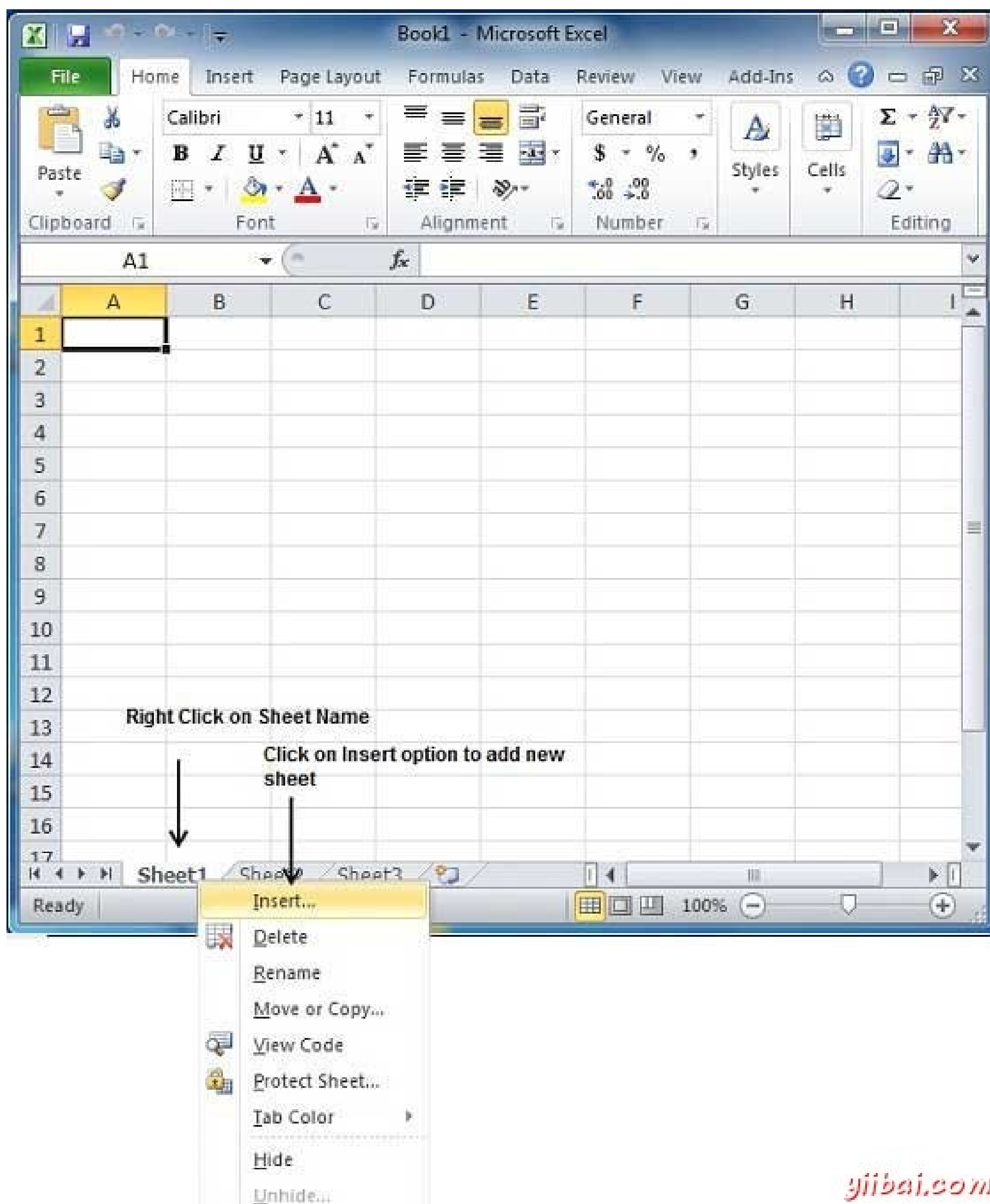
如果你的工作表是新的，它到现在还未保存，那么使用三个选项，将字显示一个对话框，让你选择一个文件夹， 并在输入情况下保存到新的工作表名称。

Excel创建工作表 - Excel教程

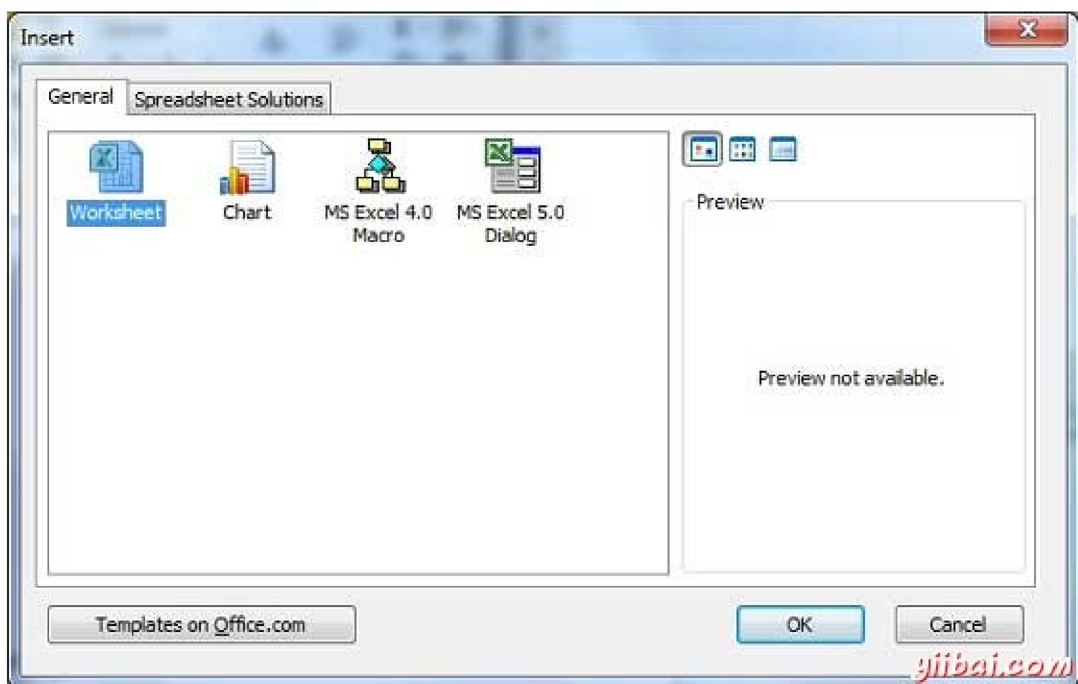
创建新的工作表

三张新的空白工作表总是在您启动Microsoft Excel时打开。但是，假设你想工作在另一个工作表上时，或者关闭已打开的工作表，并希望开始一个新的工作表开始另一个新的工作表。下面是用来创建一个新的工作表的步骤：

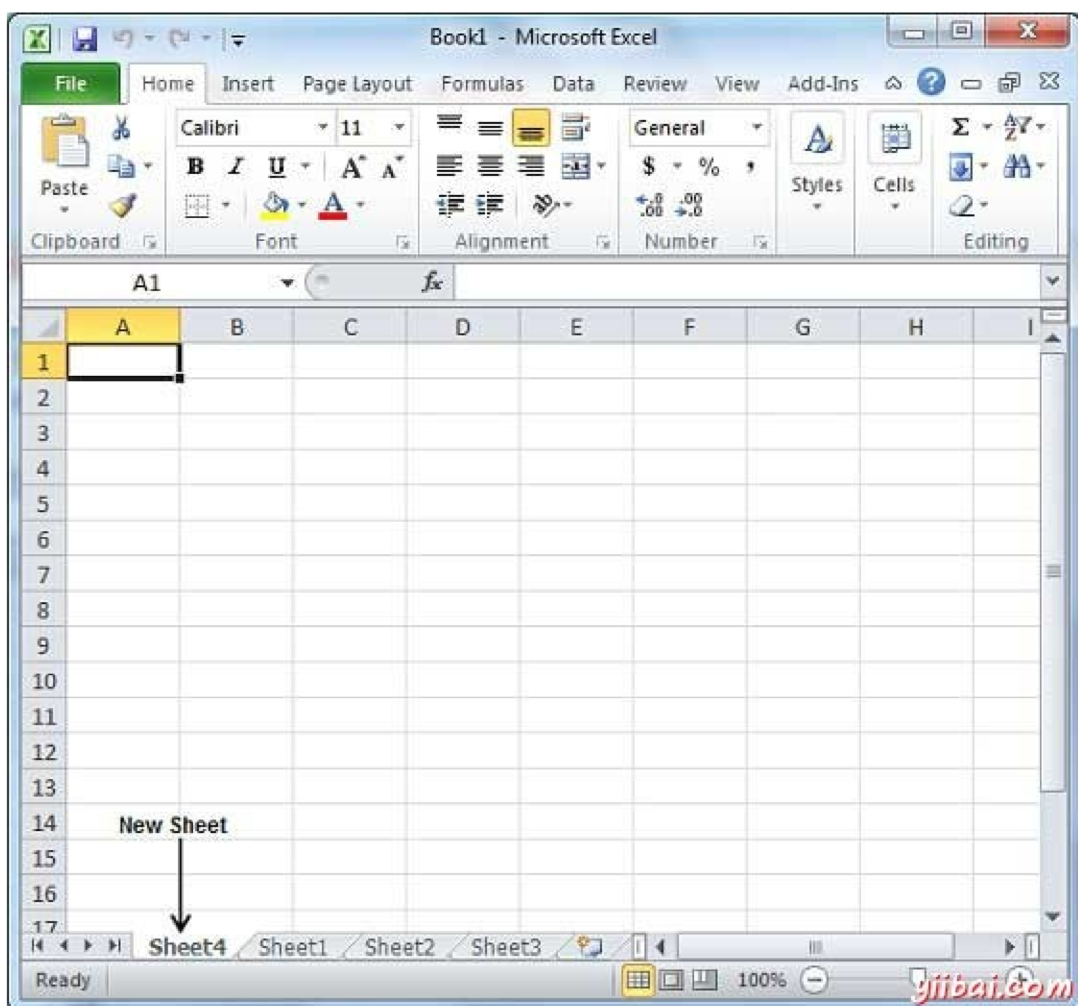
步骤(1)：右键单击工作表名称并选择插入选项。



步骤（2）：现在你会看到有选择工作表选项对话框从普通选项卡插入，选中。单击OK按钮



现在，应该有一个空白的工作表，如下图所示准备开始输入文字。



您可以随时使用一个捷径来创建一个空白工作表。尝试使用Shift + F11键，你会看到类似上面的表一个新的空白工作表打开。

Excel复制工作表 - Excel教程

复制工作表

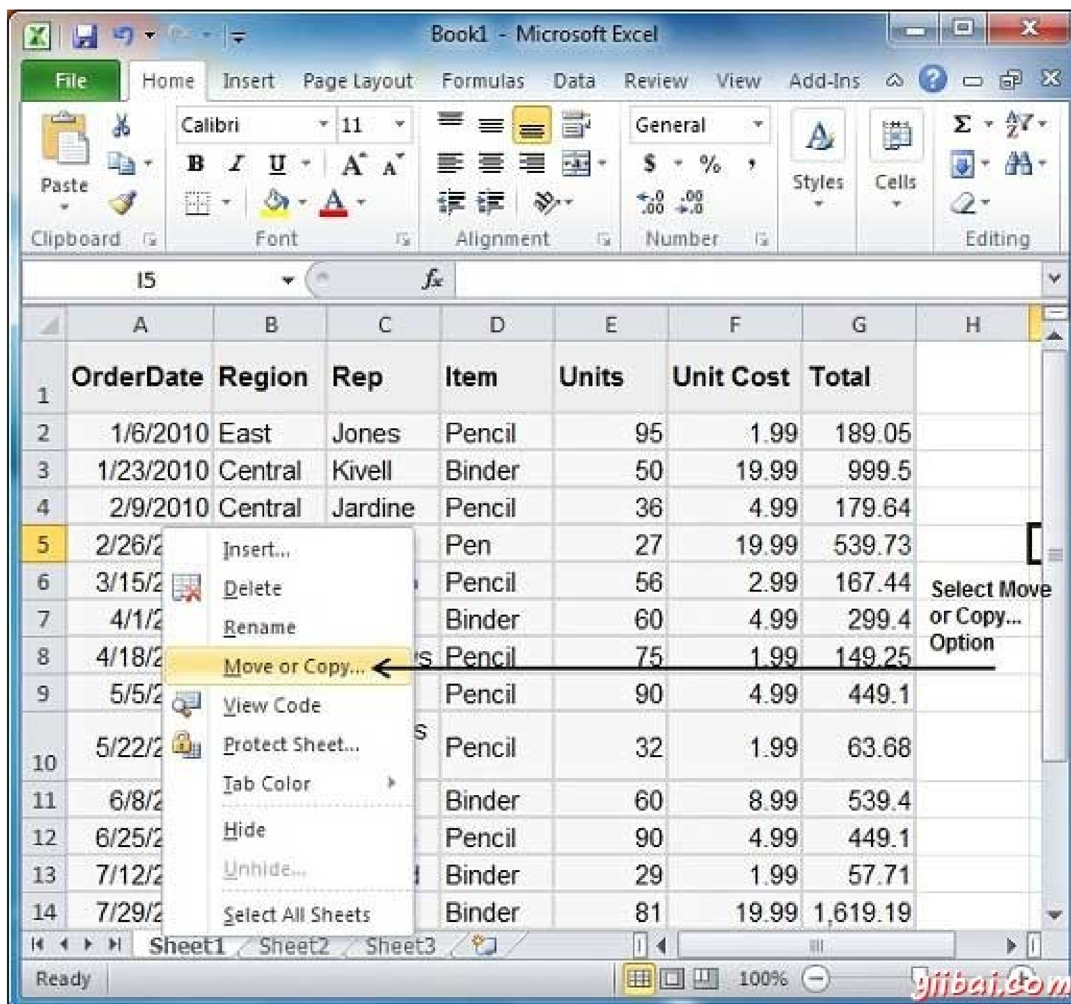
首先在创建一些示例文本之前，我们继续。 打开一个新的Excel表，然后键入一些数据。我们已经截图所示如下的样本数据。

OrderDate	Region	Rep	Item	Units	Unit Cost	Total
1/6/2010	East	Jones	Pencil	95	1.99	189.05
1/23/2010	Central	Kivell	Binder	50	19.99	999.5
2/9/2010	Central	Jardine	Pencil	36	4.99	179.64
2/26/2010	Central	Gill	Pen	27	19.99	539.73
3/15/2010	West	Sorvino	Pencil	56	2.99	167.44
4/1/2010	East	Jones	Binder	60	4.99	299.4
4/18/2010	Central	Andrews	Pencil	75	1.99	149.25
5/5/2010	Central	Jardine	Pencil	90	4.99	449.1
5/22/2010	West	Thompson	Pencil	32	1.99	63.68
6/8/2010	East	Jones	Binder	60	8.99	539.4
6/25/2010	Central	Morgan	Pencil	90	4.99	449.1
7/12/2010	East	Howard	Binder	29	1.99	57.71
7/29/2010	East	Parent	Binder	81	19.99	1,619.19
8/15/2010	East	Jones	Pencil	35	4.99	174.65

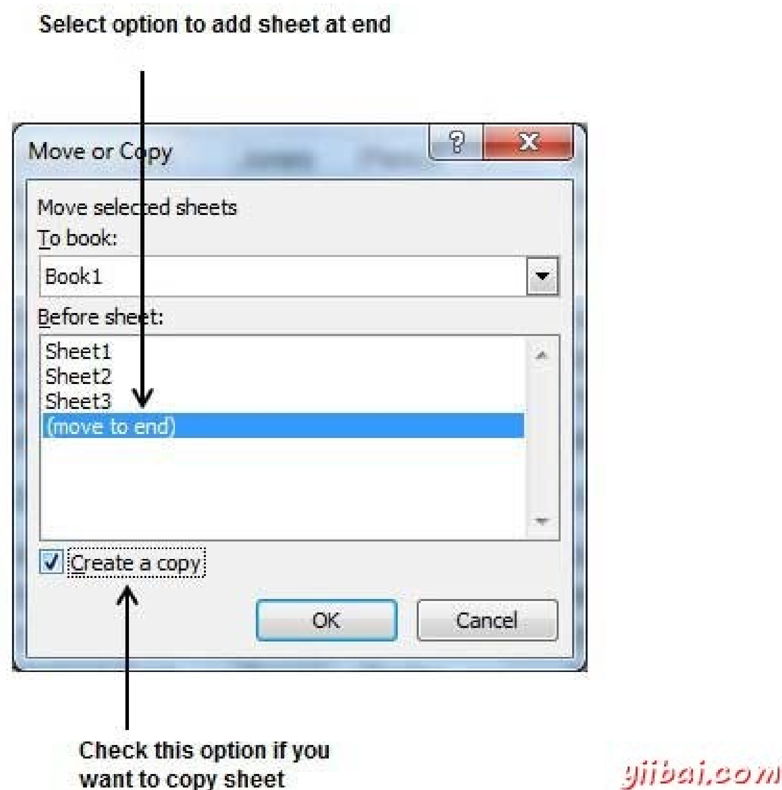
OrderDate	Region	Rep	Item	Units	Unit Cost	Total
1/6/2010	East	Jones	Pencil	95	1.99	189.05
1/23/2010	Central	Kivell	Binder	50	19.99	999.5
2/9/2010	Central	Jardine	Pencil	36	4.99	179.64
2/26/2010	Central	Gill	Pen	27	19.99	539.73
3/15/2010	West	Sorvino	Pencil	56	2.99	167.44
4/1/2010	East	Jones	Binder	60	4.99	299.4
4/18/2010	Central	Andrews	Pencil	75	1.99	149.25
5/5/2010	Central	Jardine	Pencil	90	4.99	449.1
5/22/2010	West	Thompsor	Pencil	32	1.99	63.68
6/8/2010	East	Jones	Binder	60	8.99	539.4
6/25/2010	Central	Morgan	Pencil	90	4.99	449.1
7/12/2010	East	Howard	Binder	29	1.99	57.71
7/29/2010	East	Parent	Binder	81	19.99	1,619.19
8/15/2010	East	Jones	Pencil	35	4.99	174.65

下面是复制整个工作表的步骤

步骤**(1)**：右键单击工作表名称，然后选择移动或复制选项。



步骤(2)：现在会看到移动或复制对话框，选择工作表选项在常规选项卡为选中状态。单击OK按钮



选择创建一个副本复选框以创建当前工作表的副本并在工作表的选项如(移至结束)，使新的工作表被创建。

按OK按钮

现在，你应该有一个复制工作表，如下图所示。

	A	B	C	D	E	F	G	H
1	OrderDate	Region	Rep	Item	Units	Unit Cost	Total	
2	1/6/2010	East	Jones	Pencil	95	1.99	189.05	
3	1/23/2010	Central	Kivell	Binder	50	19.99	999.5	
4	2/9/2010	Central	Jardine	Pencil	36	4.99	179.64	
5	2/26/2010	Central	Gill	Pen	27	19.99	539.73	
6	3/15/2010	West	Sorvino	Pencil	56	2.99	167.44	
7	4/1/2010	East	Jones	Binder	60	4.99	299.4	
8	4/18/2010	Central	Andrews	Pencil	75	1.99	149.25	
9	5/5/2010	Central	Jardine	Pencil	90	4.99	449.1	
10	5/22/2010	West	Thompson	Pencil	32	1.99	63.68	
11	6/8/2010	East	Jones	Binder	60	8.99	539.4	
12	6/25/2010	Central	Morgan	Pencil	90	4.99	449.1	
13	7/12/2010	East	Howard	Binder	29	1.99	57.71	
14	7/29/2010	East	Parent	Binder	81	19.99	1,619.19	
15	8/15/2010	East	Jones	Pencil	35	4.99	174.65	

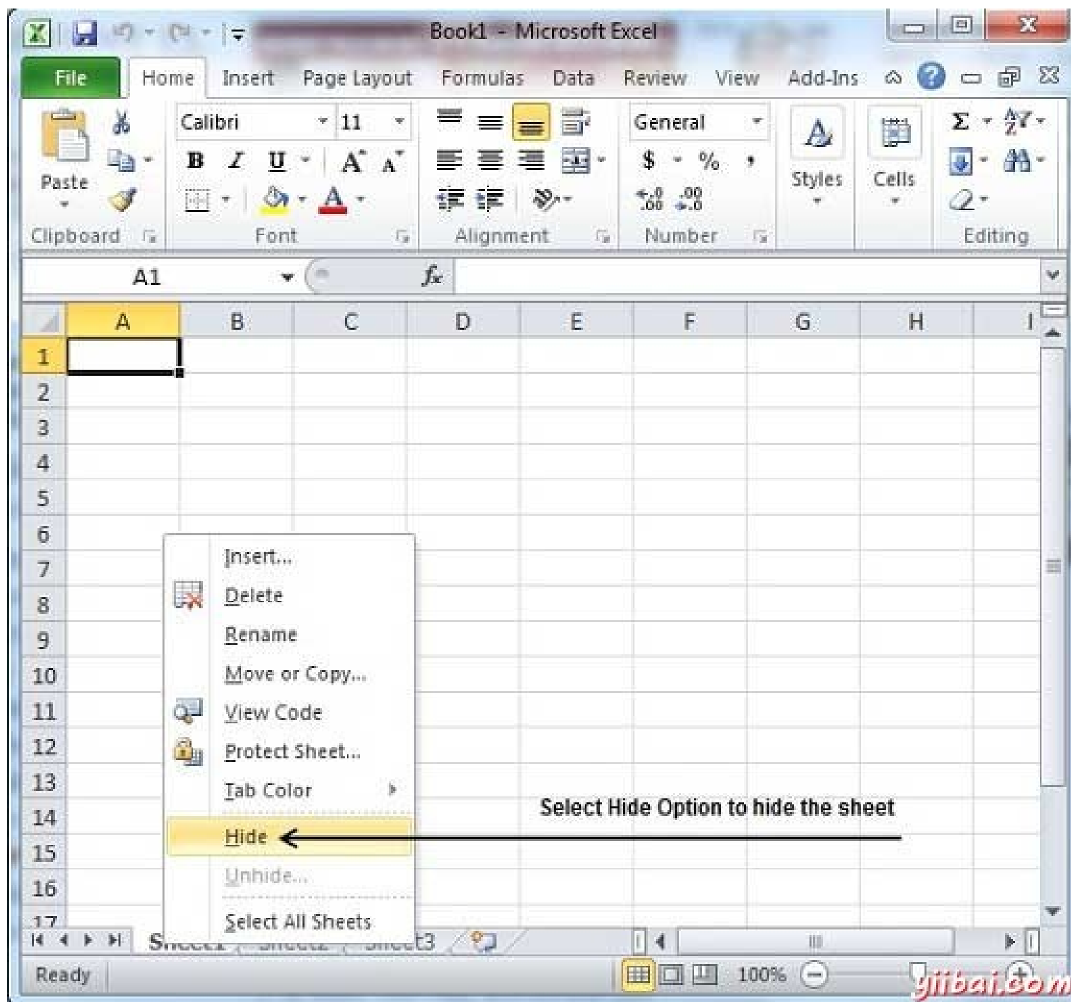
可以通过双击它重命名表名称。双击时，名称变为可编辑。输入名称为：Sheet5，然后按Tab键或回车键。

Excel隐藏工作表 - Excel教程

隐藏工作表

下面是隐藏工作表的步骤

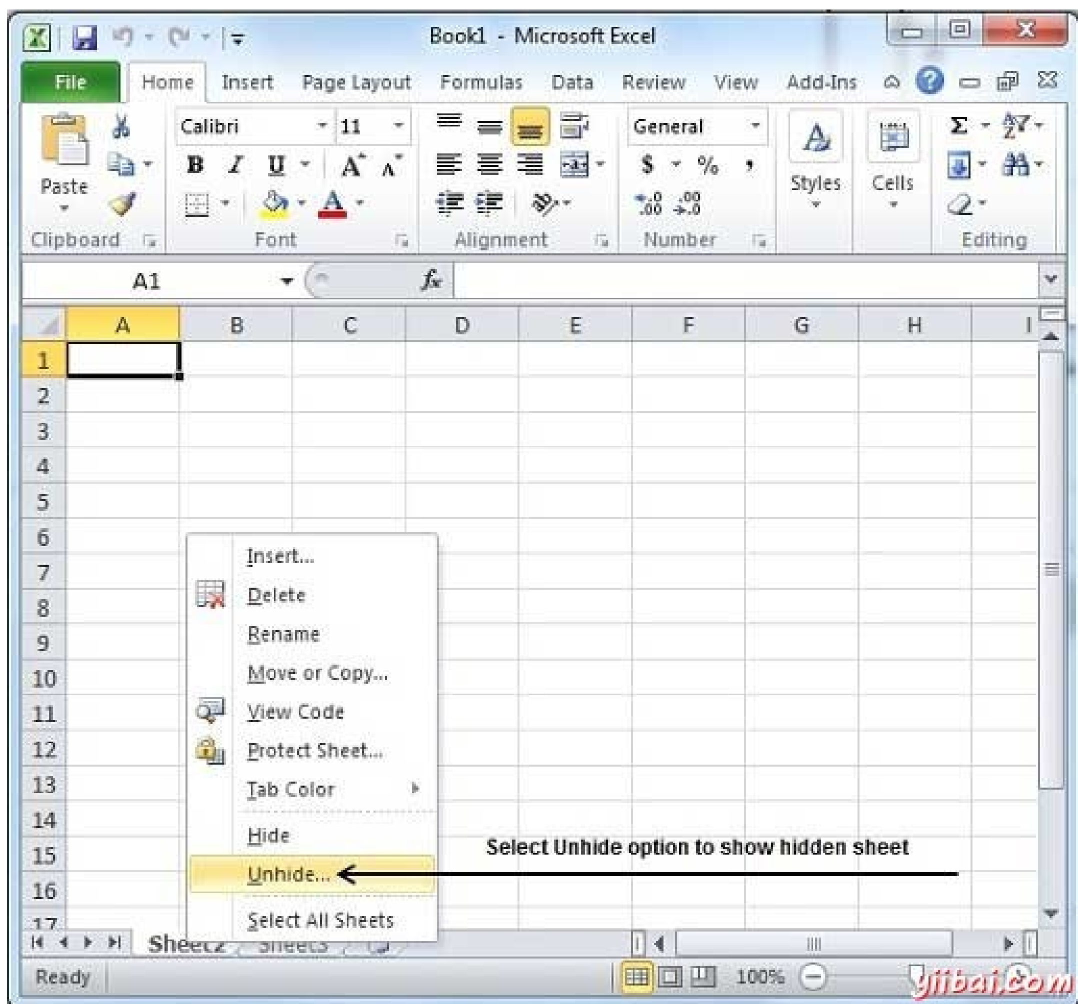
步骤(1)右键单击工作表名称并选择隐藏选项。工作表将得到隐藏。



取消隐藏工作表

下面是步骤来取消隐藏工作表

步骤（1）右键单击任何工作表名称，并选择取消隐藏...选项。



步骤（2）选择工作表名称中取消隐藏对话框中取消隐藏取消隐藏的工作表。

按OK键确认

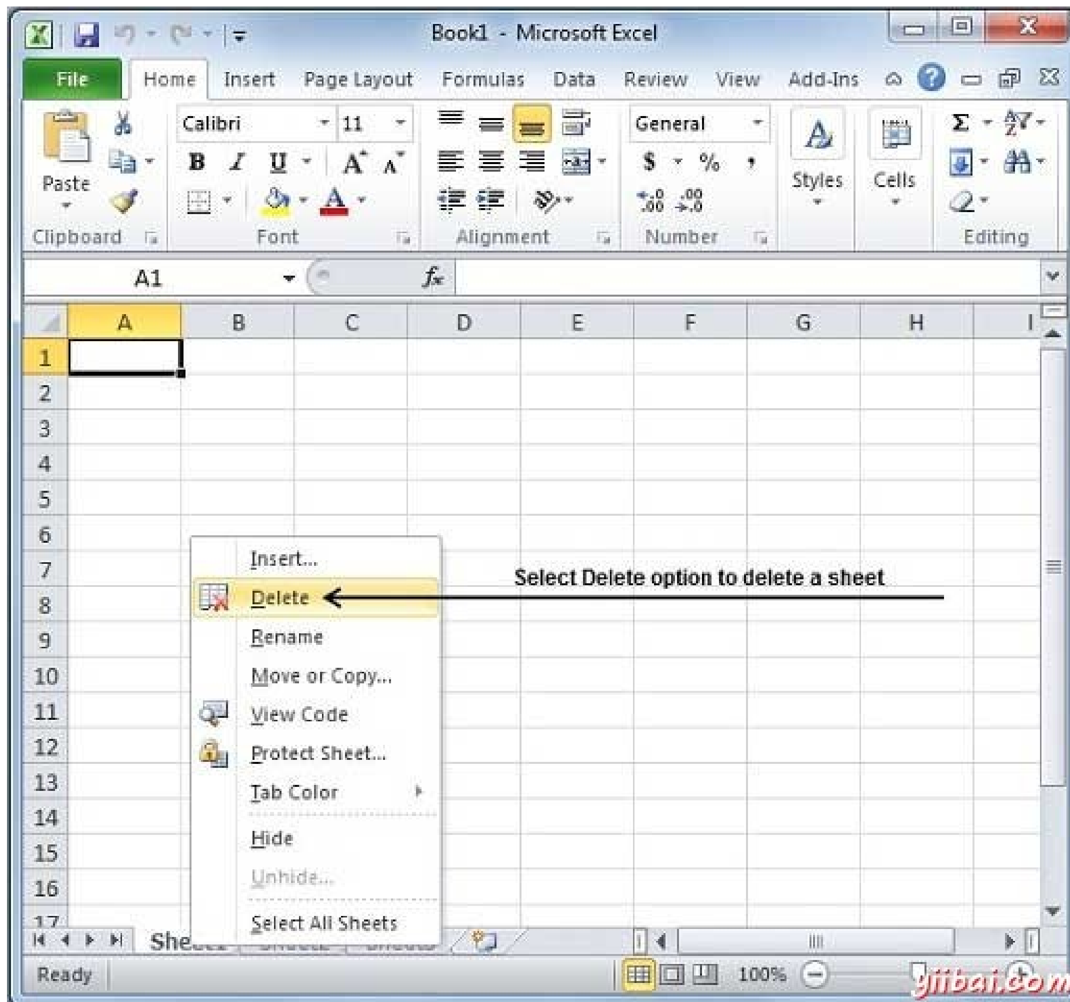
现在，你自己隐藏工作表回来。

Excel删除工作表 - Excel教程

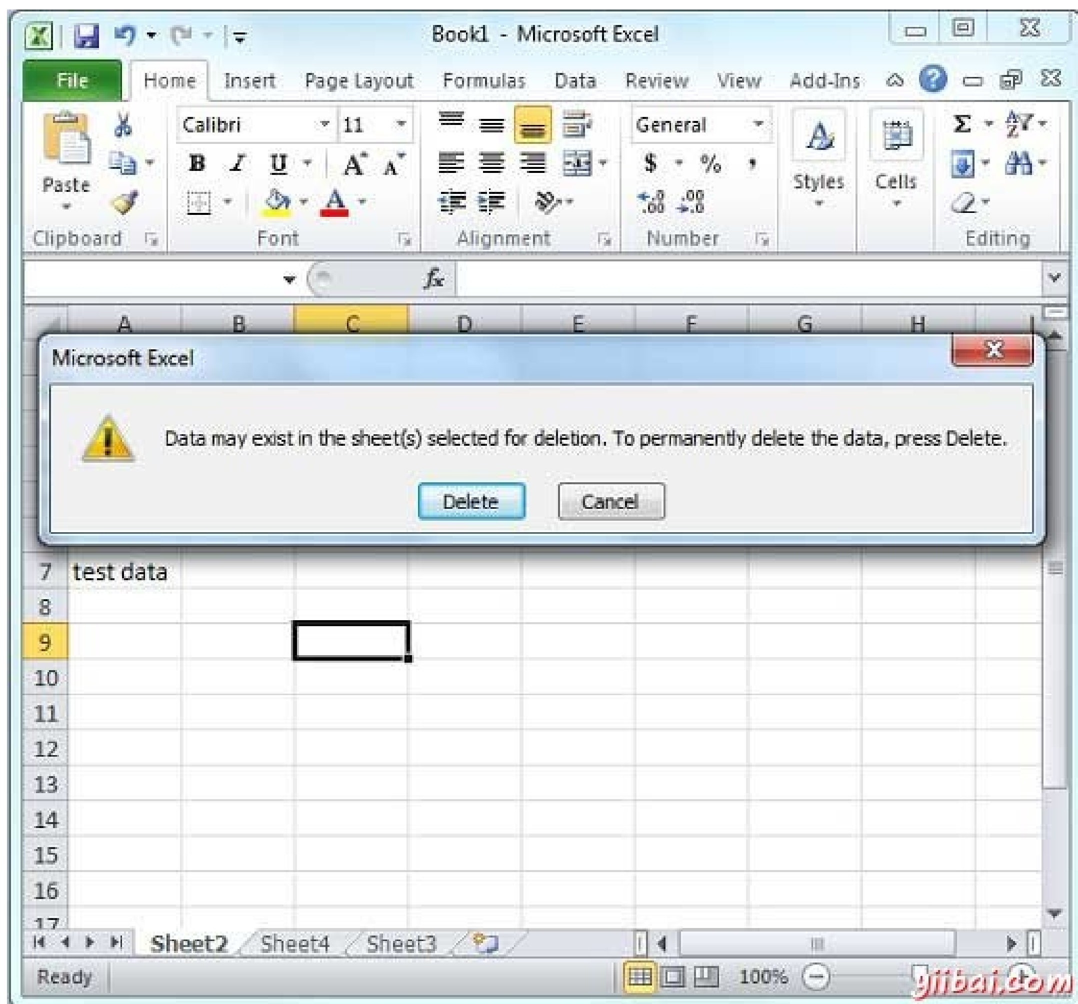
Excel删除工作表

这里是删除工作表的步骤

步骤(1)右键单击工作表名称，然后选择删除选项。



工作表将是空的，否则你会看到一条确认信息提示工作表将被删除。



步骤**(2)**按删除按钮

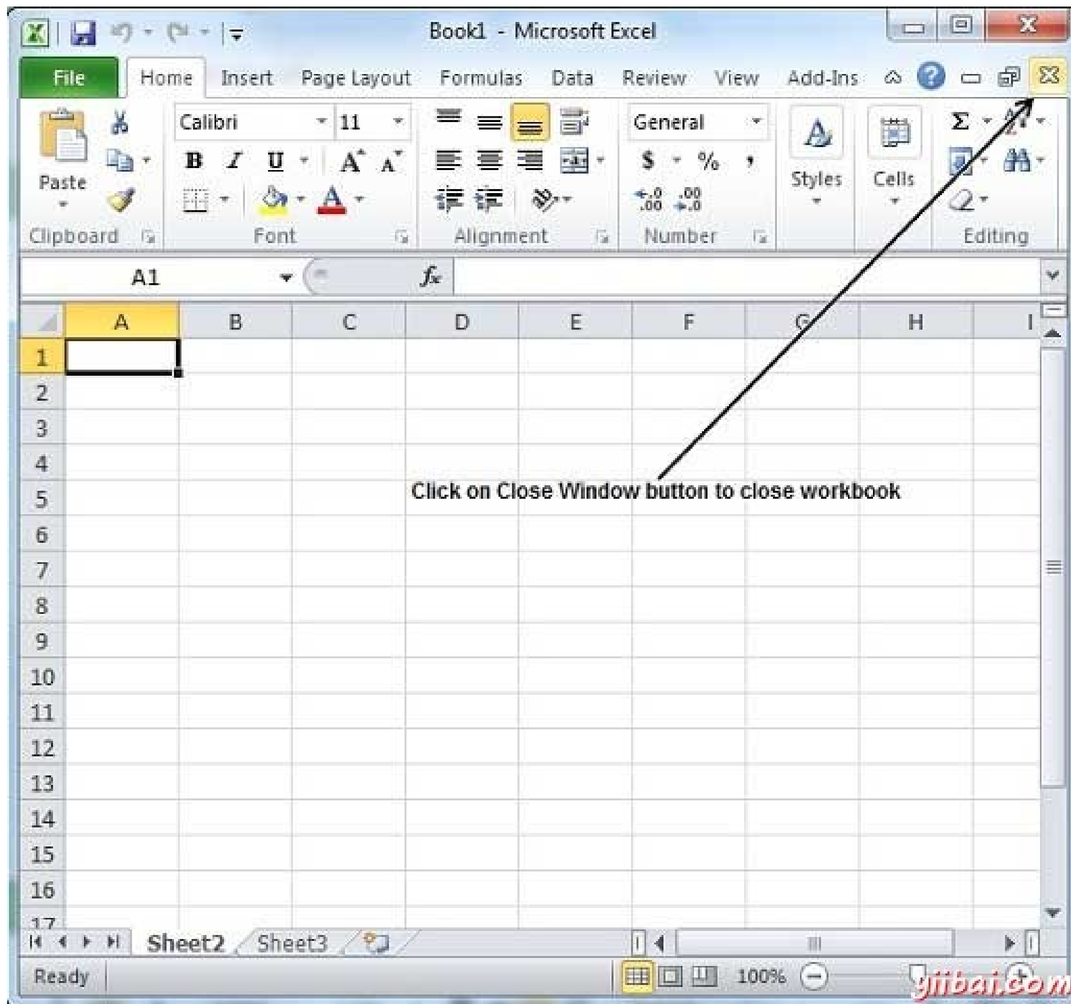
现在你的工作表将被删除。

Excel关闭工作簿 - Excel教程

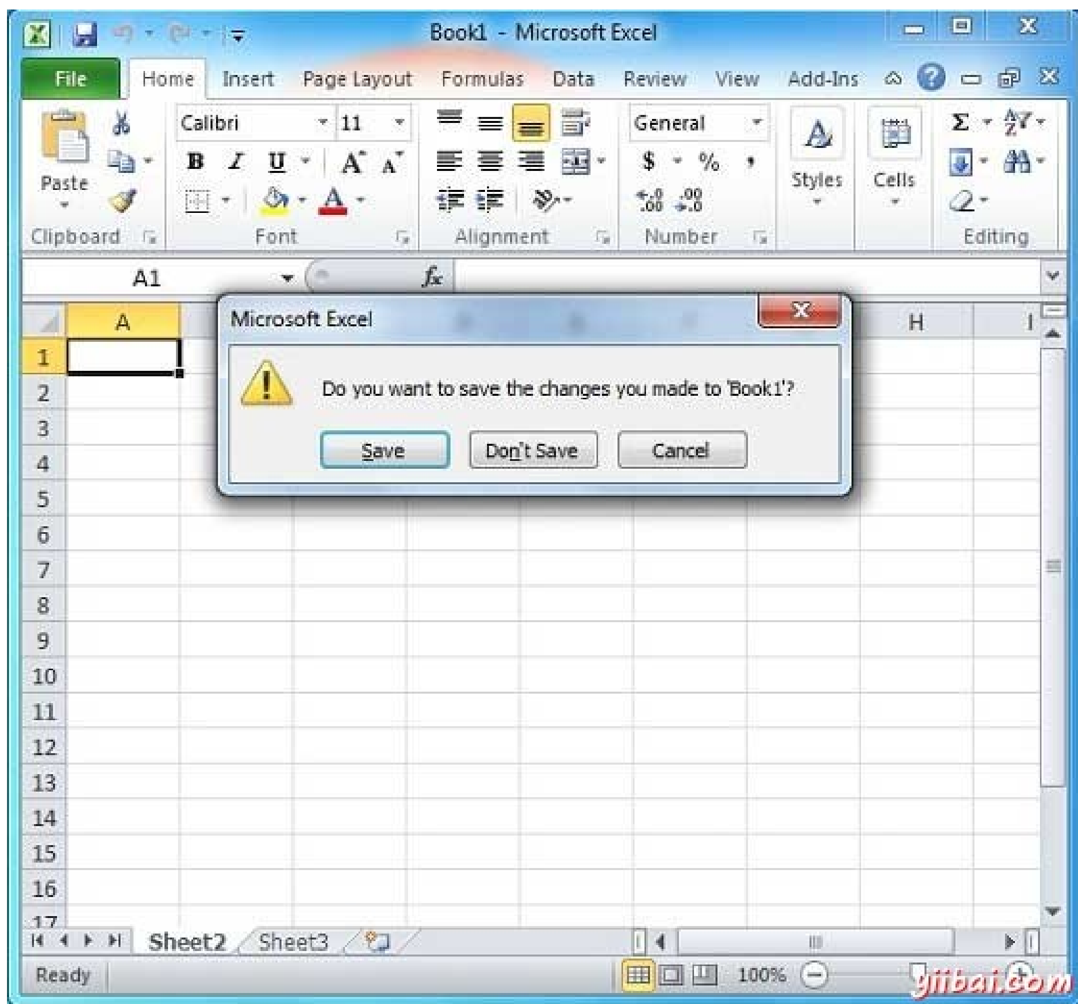
关闭工作簿

下面是关闭工作簿的步骤

步骤(1)单击关闭按钮，如下图所示。



你会看到一个保存工作簿的确认消息。



步骤**(2)**按Save按钮保存工作簿就像我们在MS Excel做 - 保存工作簿的这一章节。

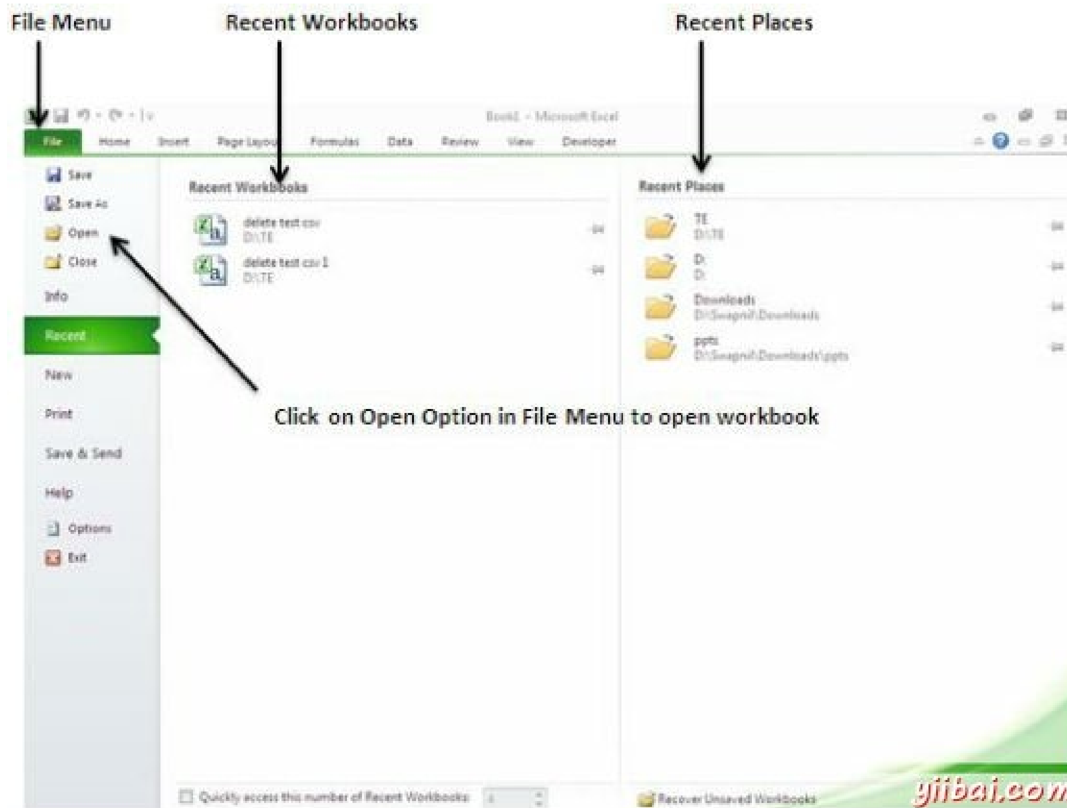
现在工作将得到保存。

Excel打开工作簿 - Excel教程

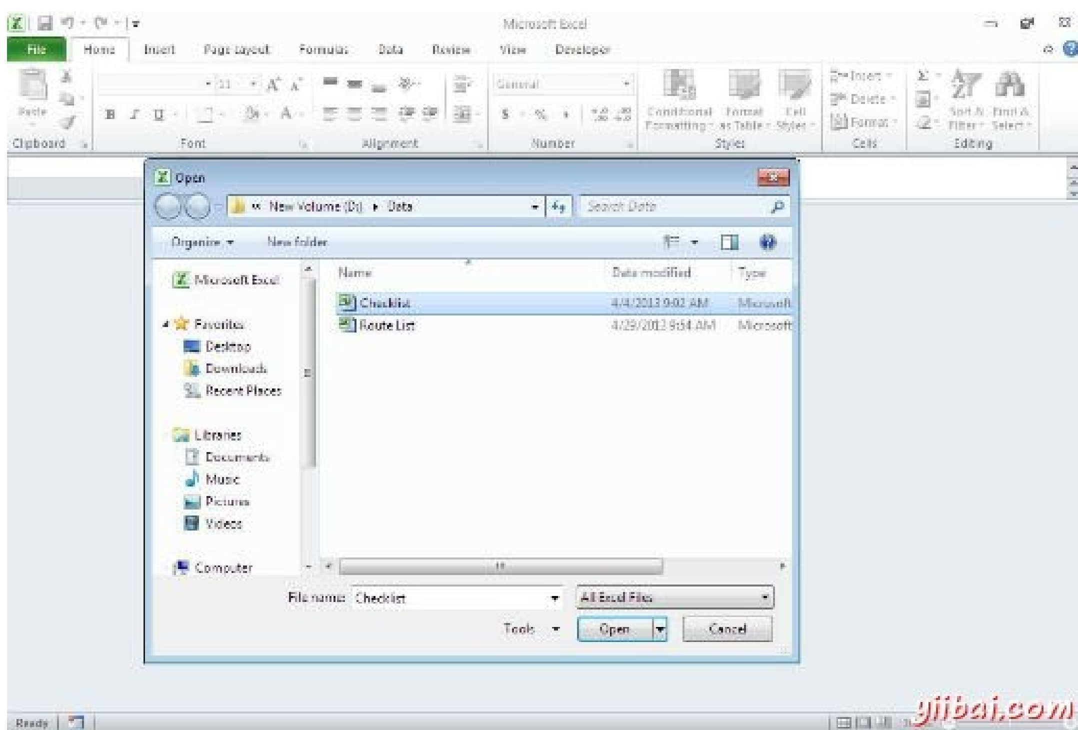
打开工作簿

让我们来看看如何打开Excel工作簿

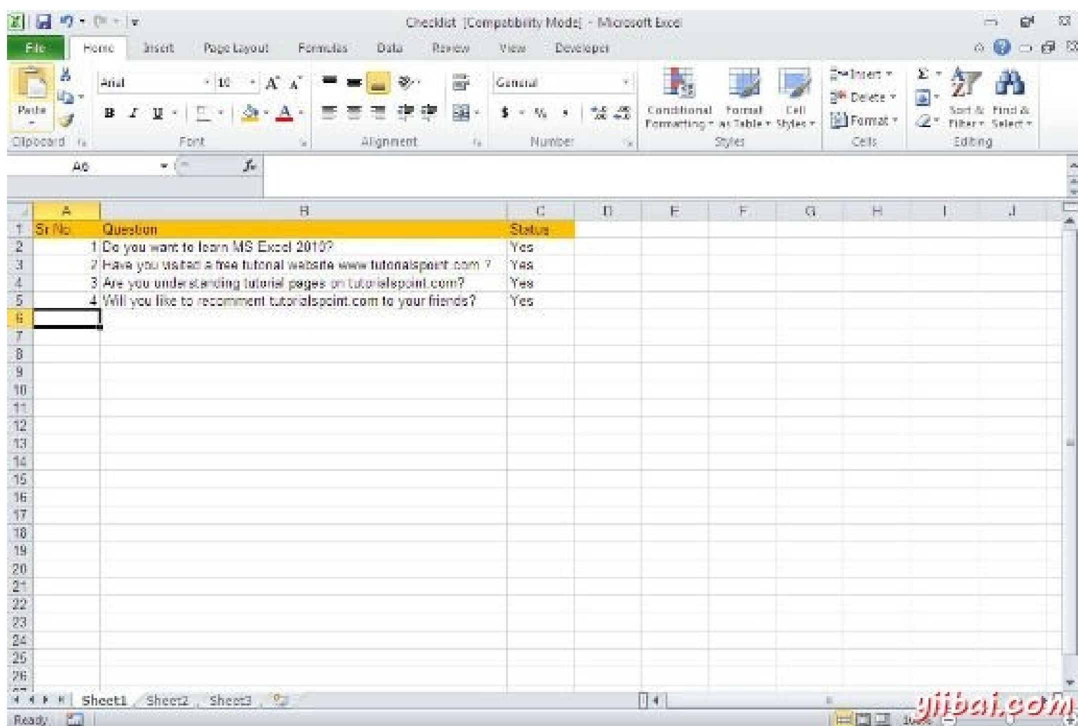
步骤**(1)**单击文件菜单，如下图所示。可以看到在文件菜单打开选项。 有两列在最近的工作簿最近的地方，你可以看到最近打开的工作簿，并从那里工作簿打开最近的地方。



步骤**(2)**点击打开选项，如下图所示将打开浏览对话框。浏览目录，找到你需要打开文件。

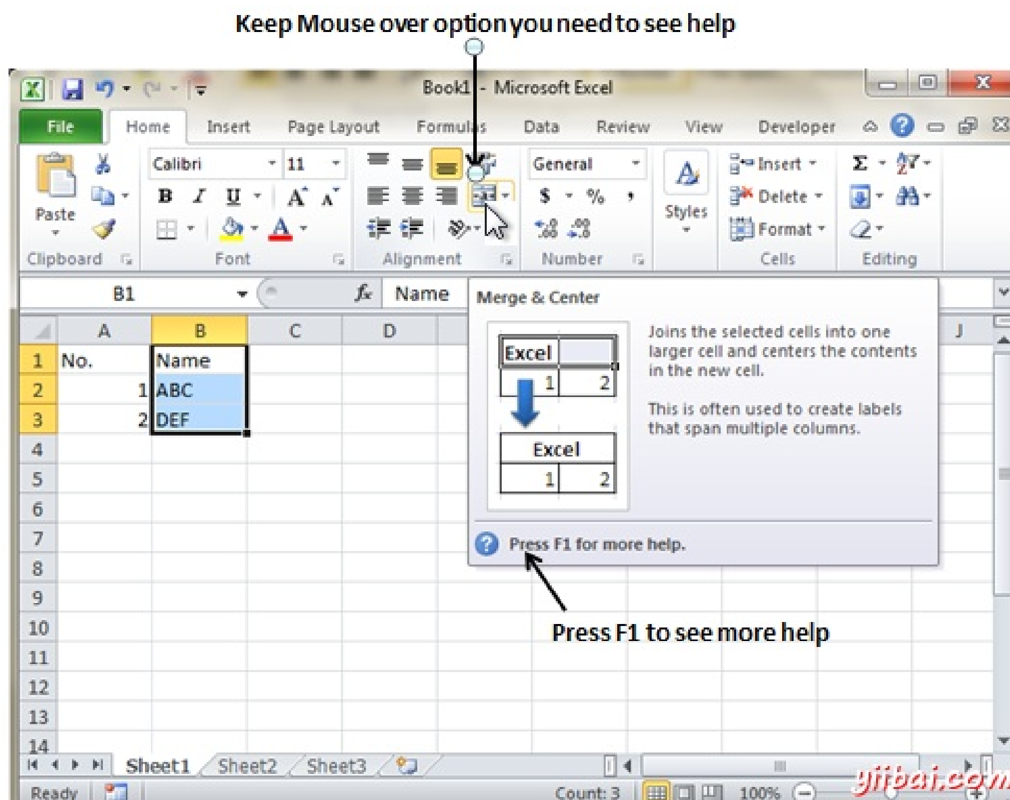


步骤**(3)**当你选择了工作簿就会被打开，如下面的工作簿：



Excel上下文帮助 - Excel教程

MS Excel中鼠标经过提供上下文相关的帮助。查看上下文特定菜单选项的帮助可将鼠标悬停在一段时间的选项。然后就可以看到上下文敏感的帮助，如下图所示



获得更多帮助

对于使用MS Excel可从微软获得更多的帮助，您可以按F1键或通过 File -> Help -> Support -> Microsoft office Help

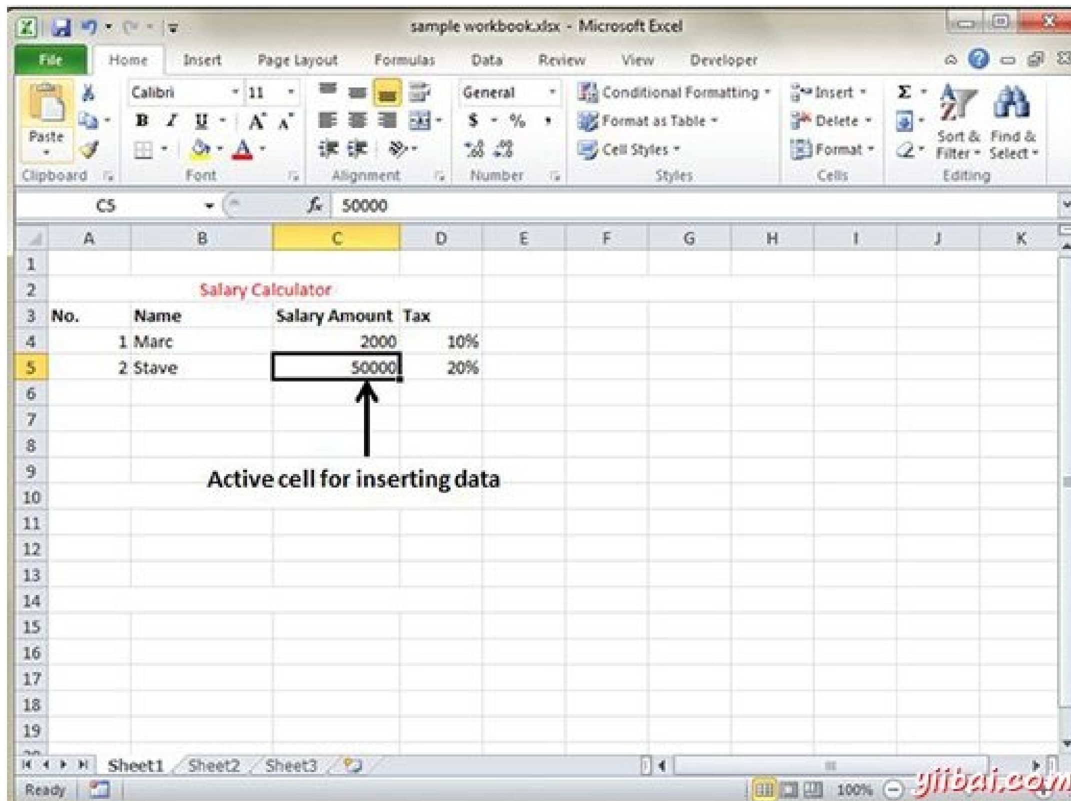

File Menu
Book1 - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Save
Save As
Open
Close
Info
Recent
New
Print
Save & Send
Help
Options
Exit
Support
Microsoft Office Help
Get help using Microsoft Office.
Tools for Working With Office
Options
Check for Updates
Microsoft office help file
Help option
Office
Product Activated
Microsoft Office Professional Plus 2010
This product contains Microsoft Access, Microsoft Excel, Microsoft SharePoint Workspace, Microsoft OneNote, Microsoft Outlook, Microsoft PowerPoint, Microsoft Publisher, Microsoft Word, Microsoft InfoPath.
Change Product Key
About Microsoft Excel
Version: 14.0.4734.1000 (32-bit)
Additional Version and Copyright Information
Part of Microsoft Office Professional Plus 2010
© 2010 Microsoft Corporation. All rights reserved.
Microsoft Customer Services and Support
Product ID: 02260-018-0000106-48057
Microsoft Software License Terms
yiibai.com

Excel插入数据 - Excel教程

在MS Excel总共有1048576*16384单元格.MS Excel单元格可以有文字，数值或公式。 MS Excel单元格中最多可以放32000个字符。

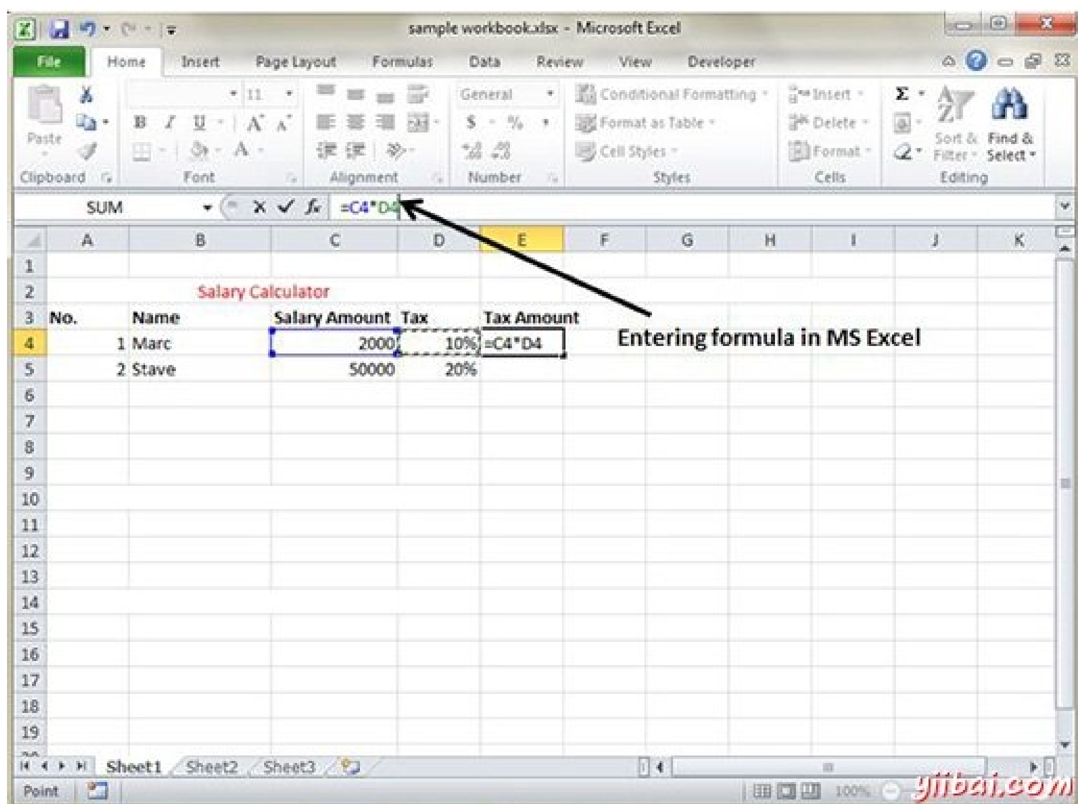
插入数据

对于插入在MS Excel中的数据只是激活单元格类型的文字或数字，然后按回车键或导航键。



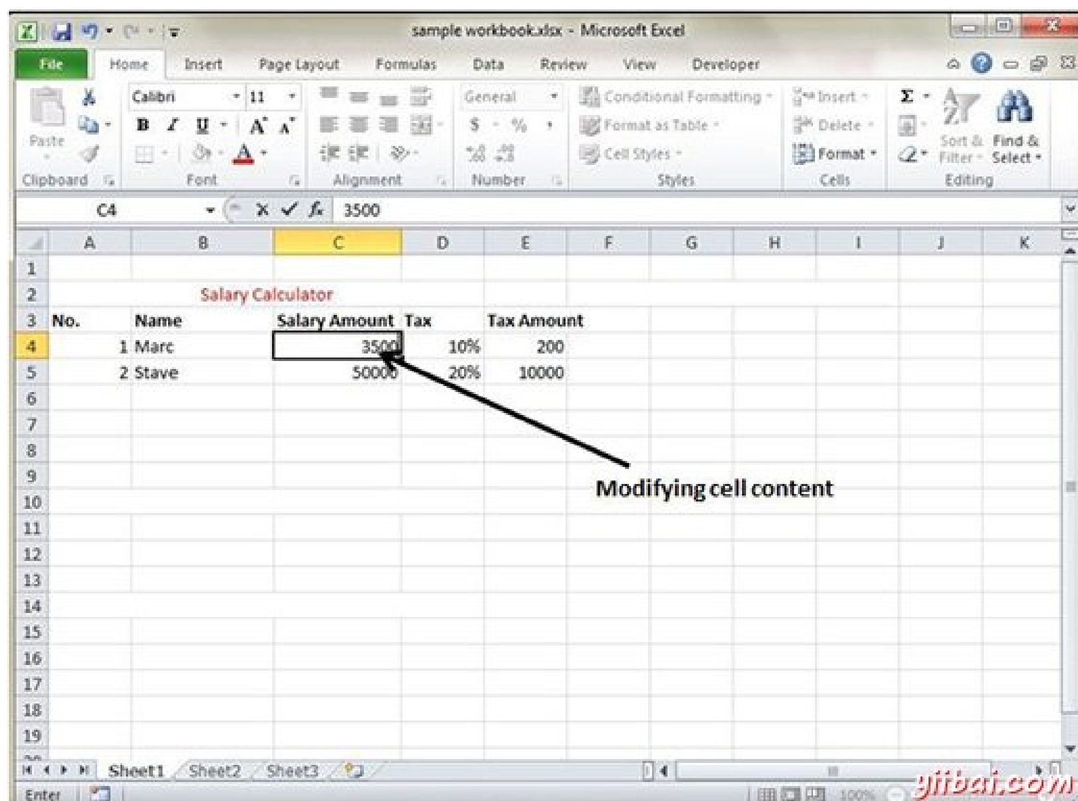
插入公式

对于在MS Excel中插入公式去公式栏，输入公式，然后按回车键或导航键。请参阅下面的屏幕截图来了解它。



修改单元格内容

修改单元格内容只是激活单元格，输入一个新值，然后按回车键或导航键看到变化。请参阅下面的屏幕截图来了解它。

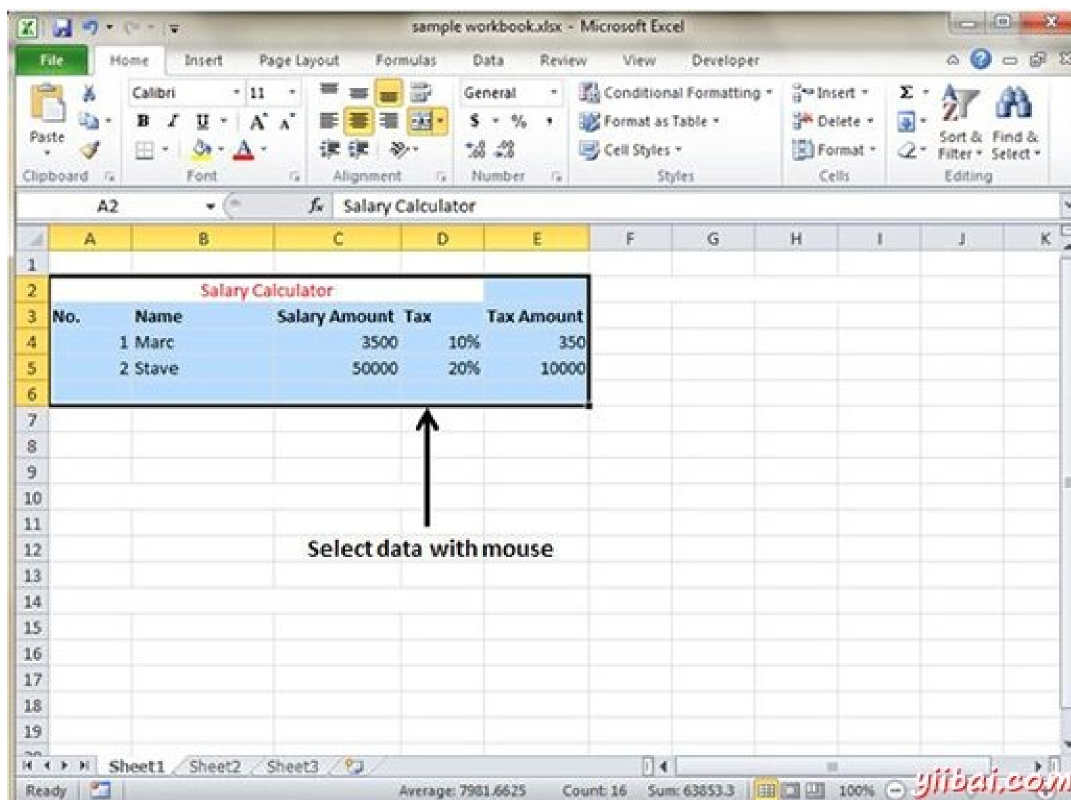


Excel选择数据 - Excel教程

MS Excel中提供了在工作表中选择数据的多方法。让我们来看看这些方法。

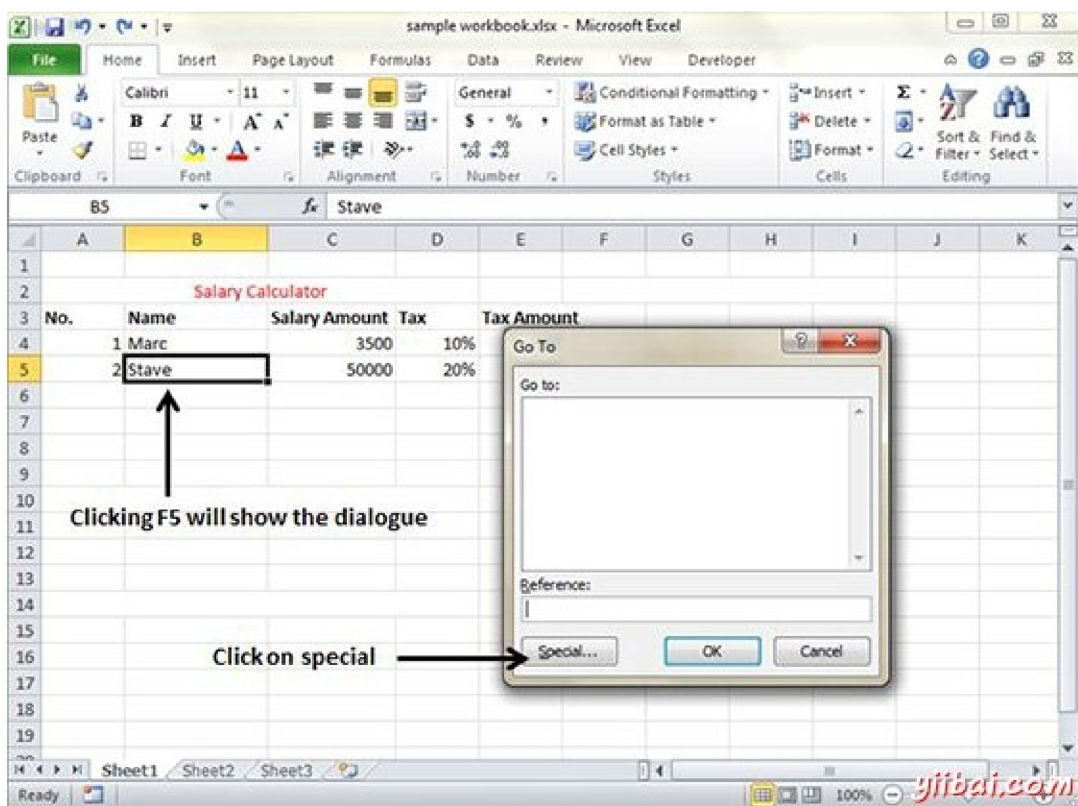
使用鼠标选择

将鼠标拖动到要选择的数据。它将选择这些单元格如下所示。

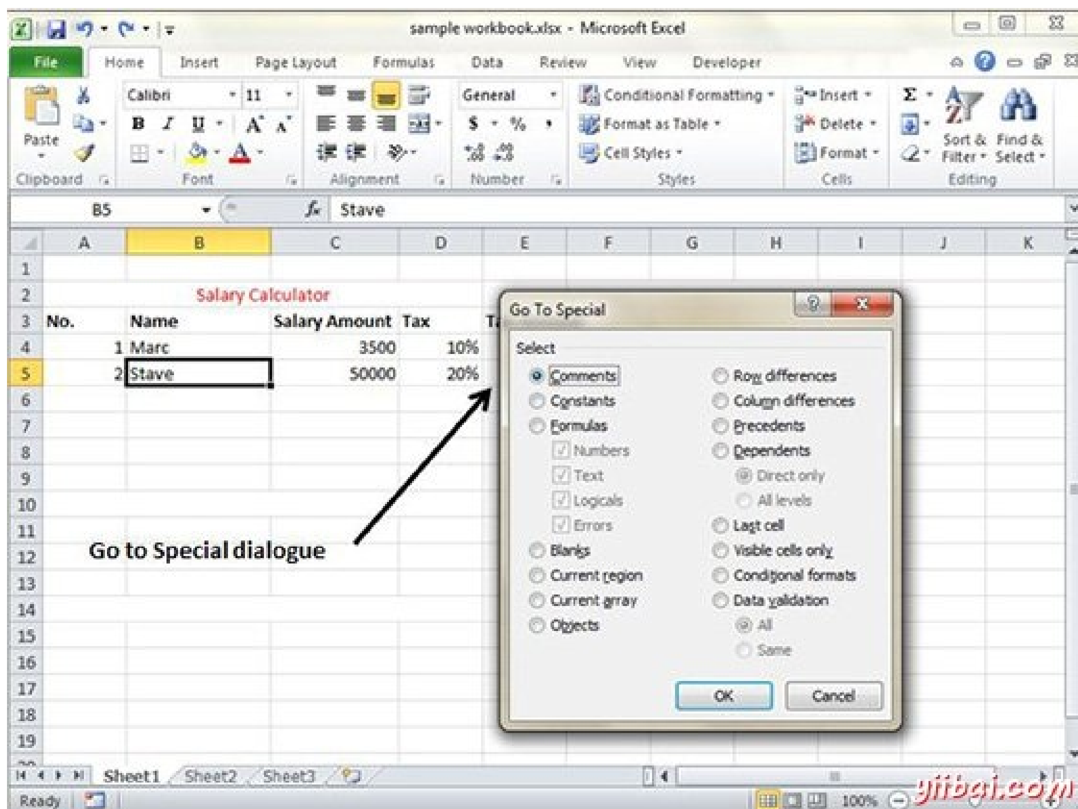


使用特殊选择

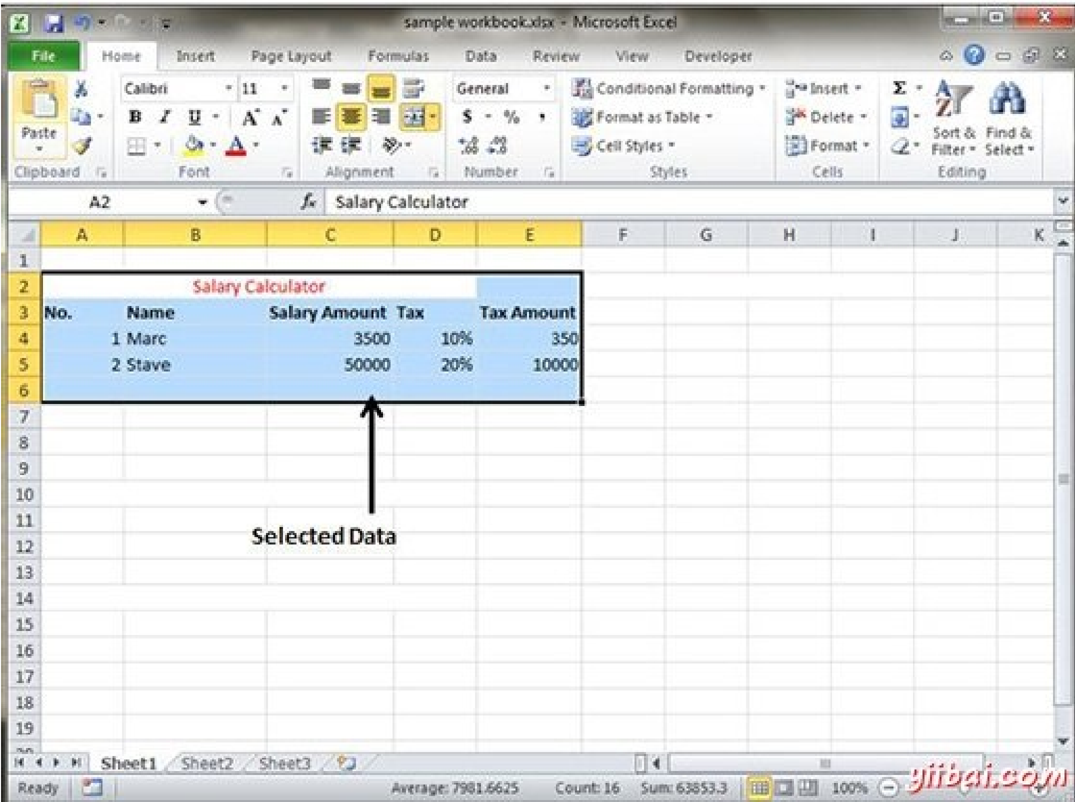
如果您想选择特定区域，选择该区域的单元格。按F5将显示如下的对话。



单击特殊按钮，看看下面的对话。从单选按钮选择当前区域。 点击OK，看当前区域选择。



可以看到在下面屏幕上的数据被选为当前区域。

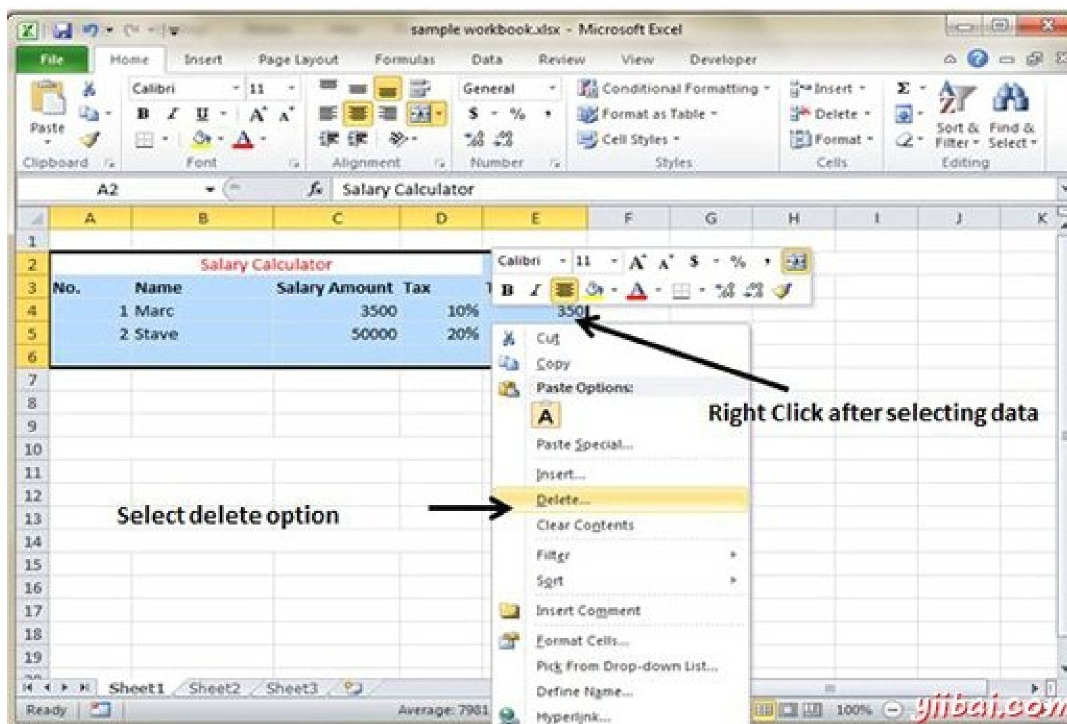

sample workbook.xlsx - Microsoft Excel
Salary Calculator
No.
Name
Salary Amount
Tax
Tax Amount
1 Marc
3500
10%
350
2 Stave
50000
20%
10000
Selected Data
Sheet1
Sheet2
Sheet3
Average: 7981.6625
Count: 16
Sum: 63853.3

Excel删除数据 - Excel教程

MS Excel提供了删除的表数据的各种方法。让我们来看看这些方法。

使用鼠标删除

选择要删除的数据。右键单击工作表上。选择删除选项，它会删除数据。

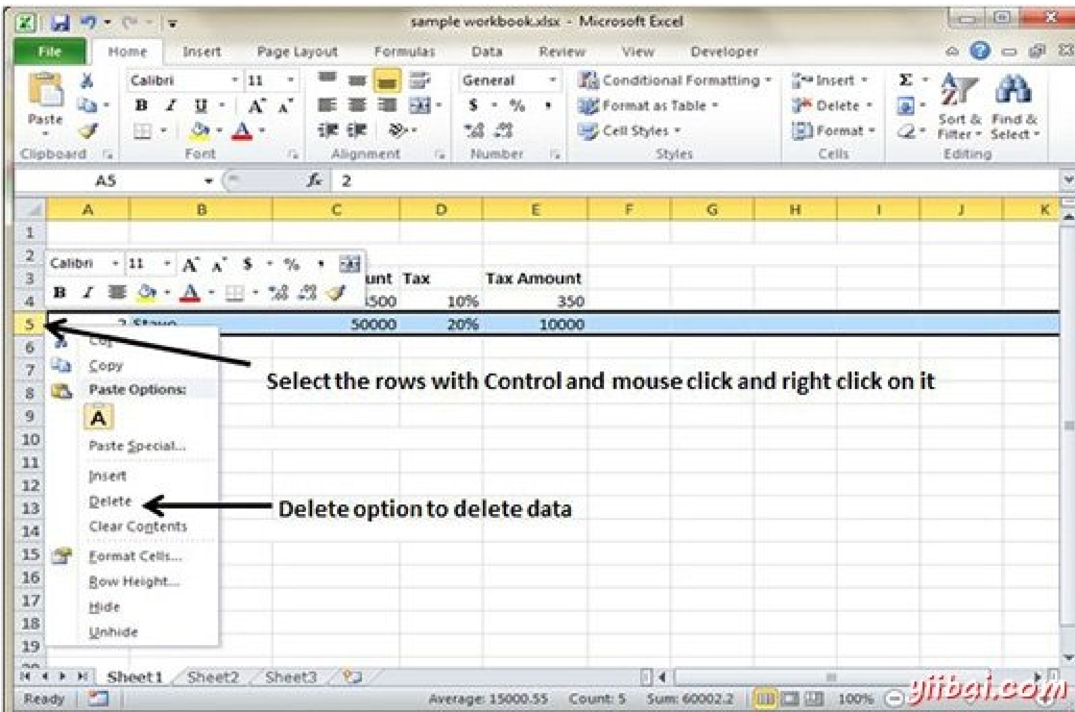


使用Delete键删除

选择要删除的数据。按键盘的Delete按钮，它会删除数据。

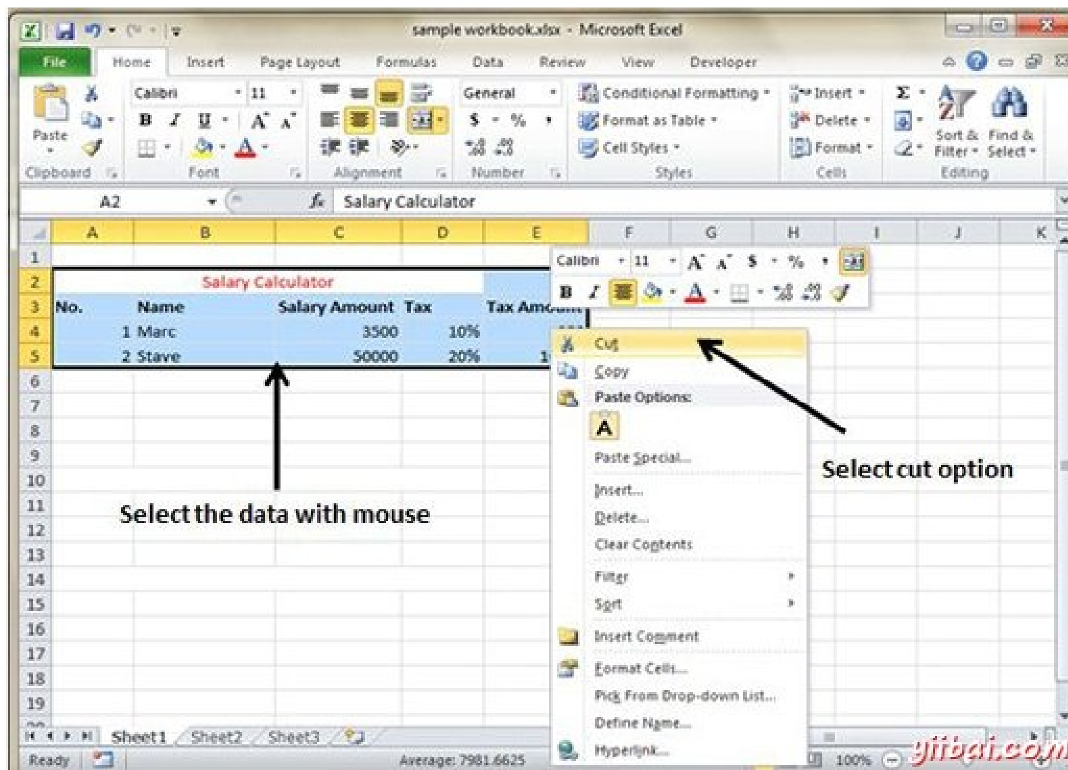
选择删除的行

选择您想要的鼠标点击+控制键来删除行。然后右键单击它会显示各种选项。选择删除选项删除选定的行。

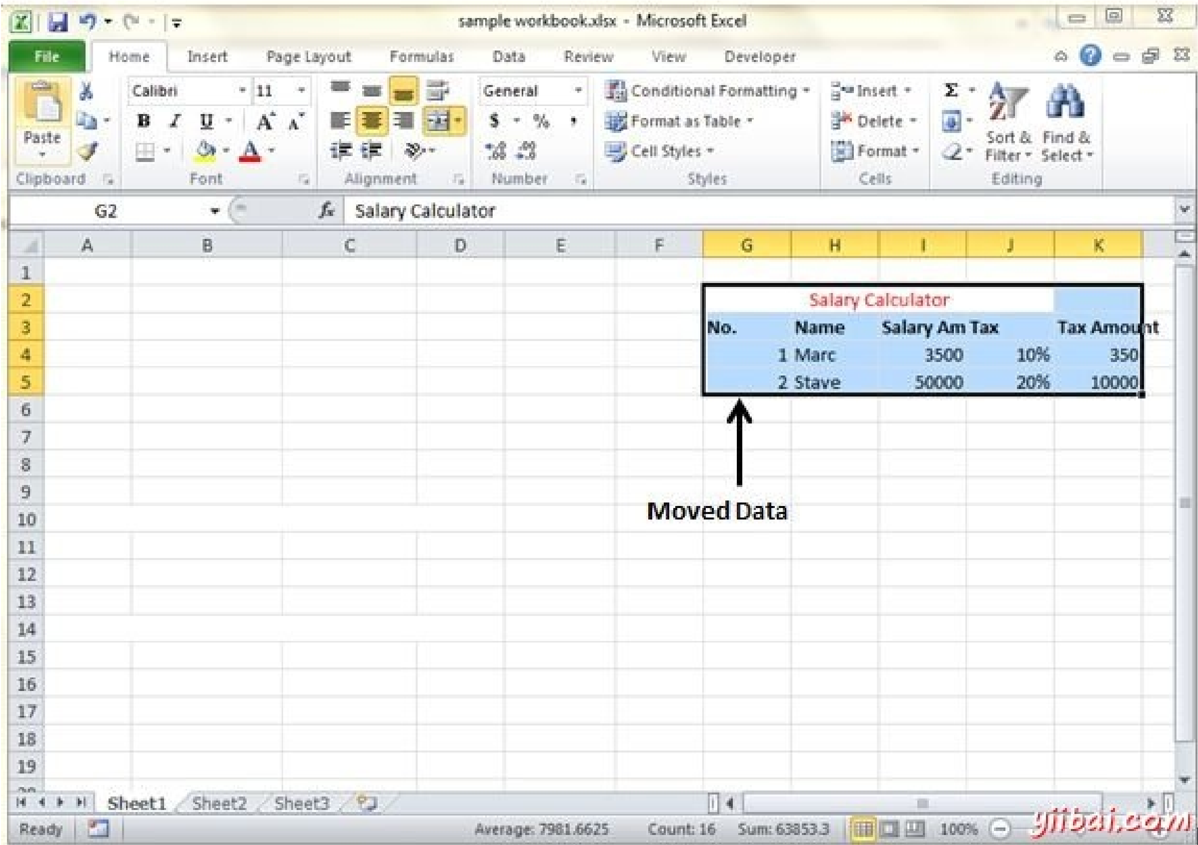

sample workbook.xlsx - Microsoft Excel
Select the rows with Control and mouse click and right click on it
Delete option to delete data

Excel移动数据 - Excel教程

让我们来看看如何使用MS Excel移动数据。步骤**(1)**选择要移动的数据。右键点击它。选择剪裁选项。



步骤**(2)**选择第一个单元格，你要移动的数据。右键点击它，并粘贴数据。你可以看到，现在移动的数据。


sample workbook.xlsx - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
G2
Salary Calculator
Salary Calculator
No.
Name
Salary Am Tax
Tax Amount
1 Marc
3500
10%
350
2 Stave
50000
20%
10000
Moved Data
Sheet1
Sheet2
Sheet3
Ready
Average: 7981.6625
Count: 16
Sum: 63853.3
100%

Excel行和列 - Excel教程

行和列的基础知识

MS Excel在表格格式是由行和列组成。

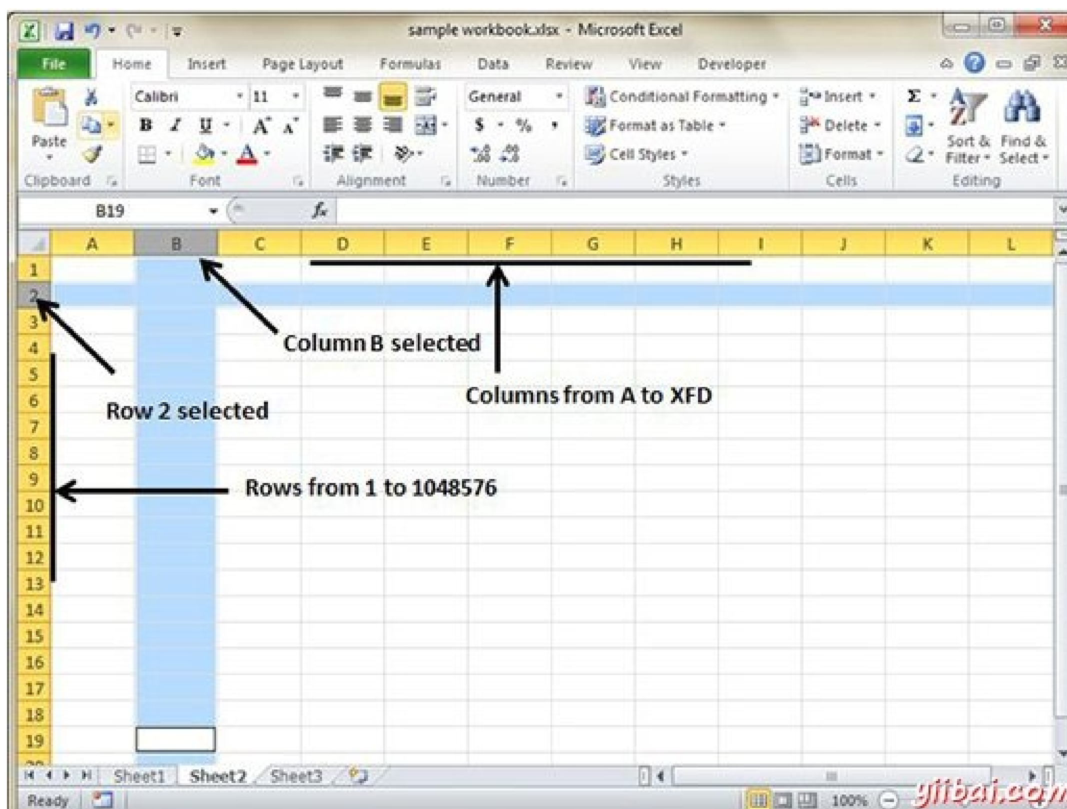
- 行为水平方向，列为垂直方向。
- 每一行确定了垂直地在工作表的左侧行号。
- 每列由在水平方向页的顶端的列标题确定

对于MS Excel 2010的行号范围从1到104857共1048576行，列的范围从A到XFD共16384列

导航行和列

让我们来看看如何移动到最后一行或最后一列。

- 您可以通过点击Ctrl +向下箭头导航到最后一行。
- 您可以通过点击控制+右箭头导航到最后一列。



单元格简介

行和列的交叉点被称为单元格。

单元格是确定与列标题和行号的组合。

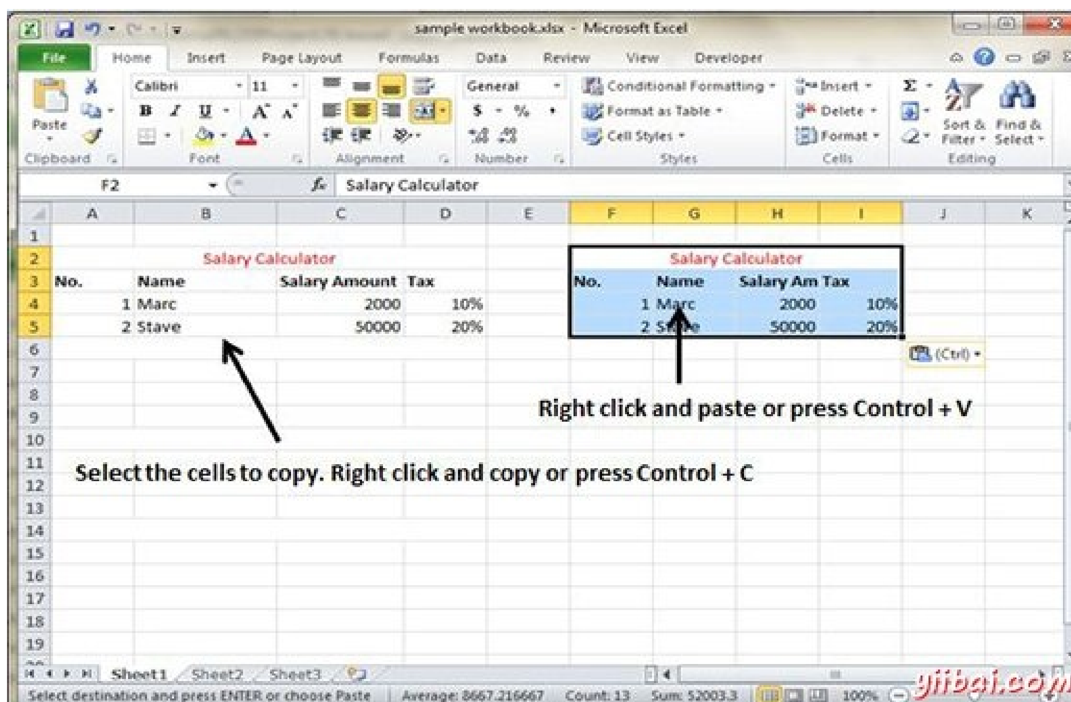
例如：A1, A2

Excel复制和粘贴 - Excel教程

MS Excel提供以不同的方式复制粘贴选项。复制粘贴的最简单的方法如下。

复制粘贴

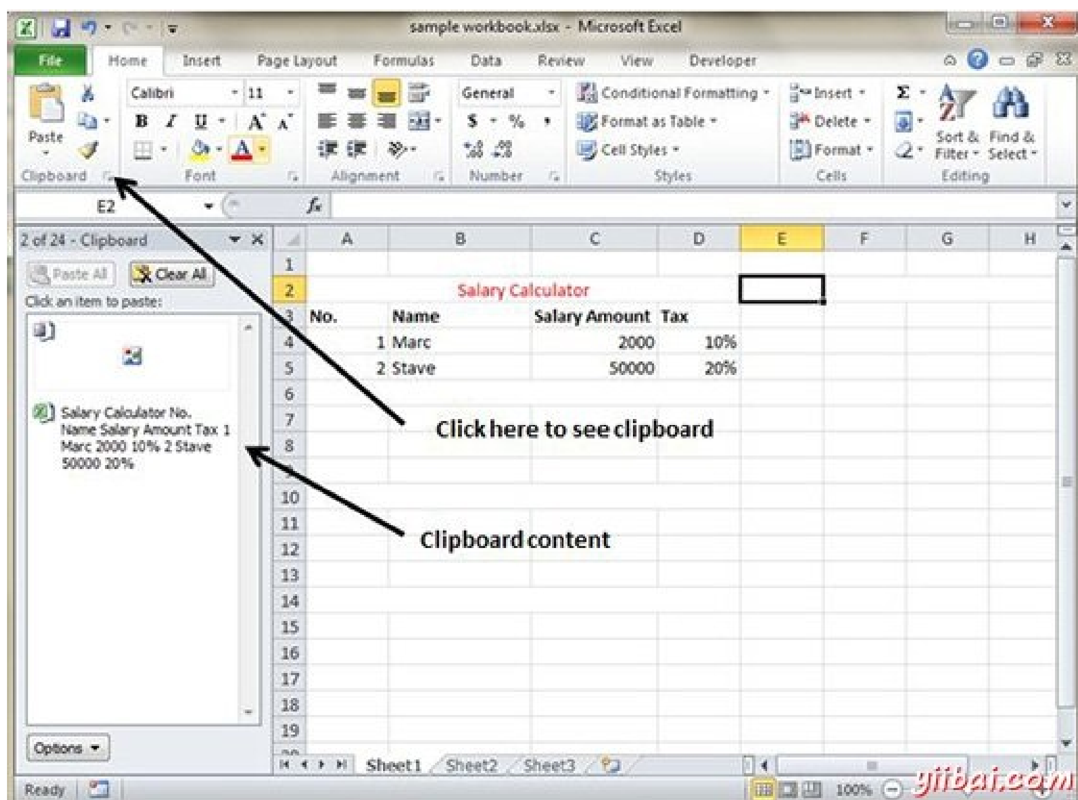
- 要做到复制粘贴只需选择要复制的单元格。选择后点击鼠标右键或者按下Control+ C.复制选项
- 选择单元格，需要粘贴复制的内容。右键单击并选择粘贴选项或按Control + V.



在这种情况下，MS Excel将复制的一切数值，公式，格式，注释和验证。MS Excel将覆盖有粘贴的内容。如果你想撤消请按Control + Z

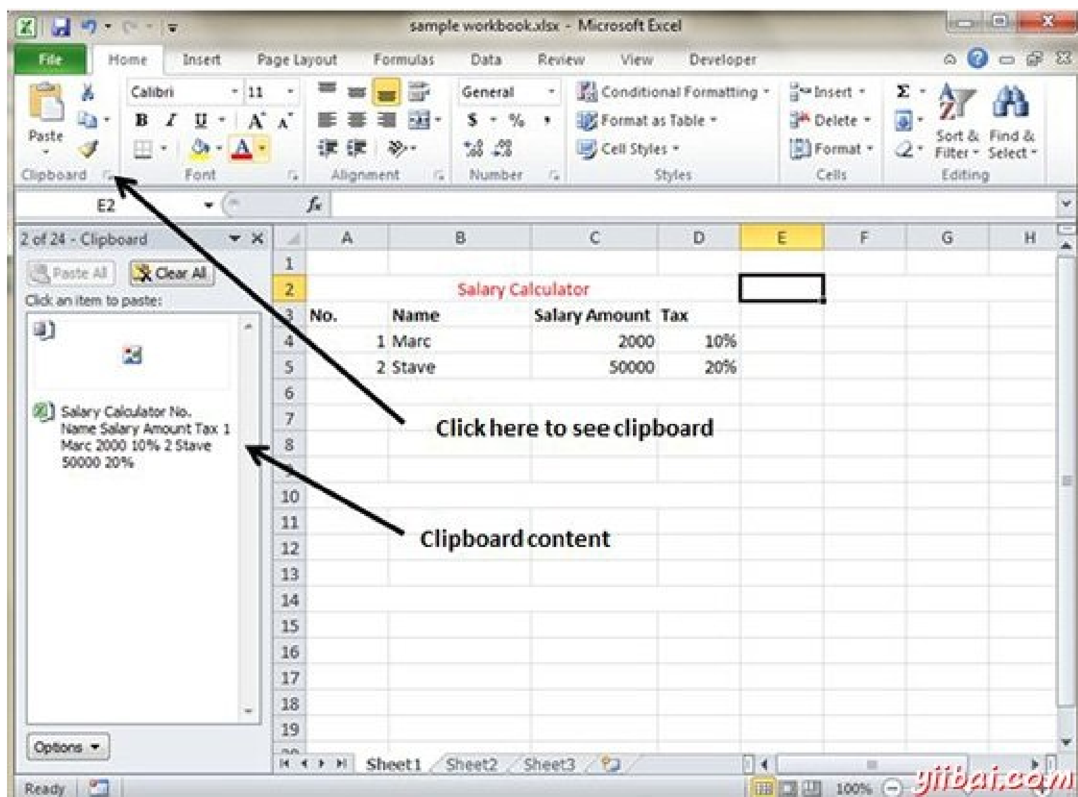
复制粘贴使用Office剪贴板

当你在MS Excel的数据复制它并放入Windows和Office剪贴板内容。您可以通过查看剪贴板中的内容 Home -> Clipboard.查看剪贴板中的内容。选择单元格并粘贴。单击粘贴，粘贴内容。



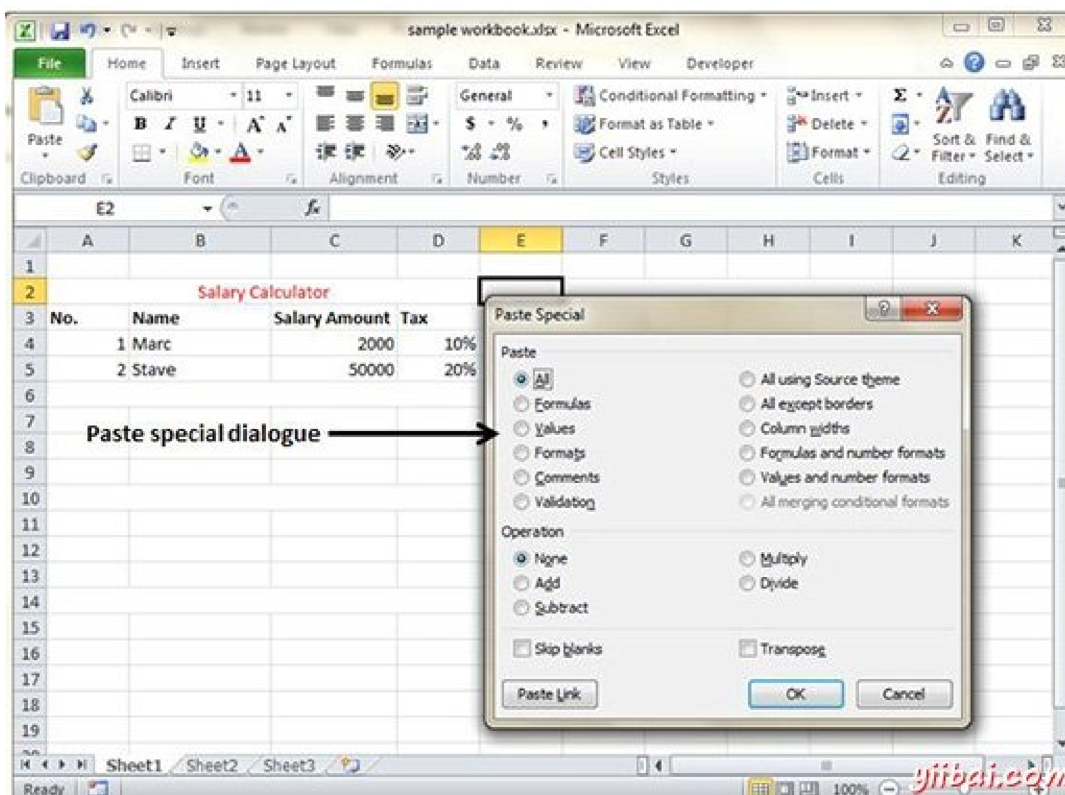
以特殊的方式复制粘贴

你可能不希望复制一切，在某些情况下只复制值，例如或要复制到单元格格式化。选择粘贴特殊选项，如下图所示。



下面是以选择性粘贴的一些选择。

- 全部: 从Windows剪贴板粘贴单元格的内容，格式和数据验证。
- 公式：粘贴公式，而不是格式化。
- 值: 只粘贴数值而不是公式。
- 格式: 粘贴源范围仅格式。
- 注释: 粘贴相应单元注释。
- 验证: 在单元格提供验证。
- 所有使用源主题：粘贴公式和所有格式。
- 所有的边框除外：粘贴一切，除了出现在源范围的边界。
- 列宽: 粘贴公式并复制复制的单元格的列宽。
- 公式和数字格式：只粘贴公式和数字格式。
- 值和数字格式: 粘贴公式的结果，加上数值
- 合并条件格式：当复制的单元格包含条件格式此图标才会显示。当点击它融合了在目标范围内的任何条件格式复制的条件格式。
- 移调: 改变的范围内复制的方向。行成为列和列将成为行。任何公式复制的范围内进行调整，以使它们正常工作时换位。



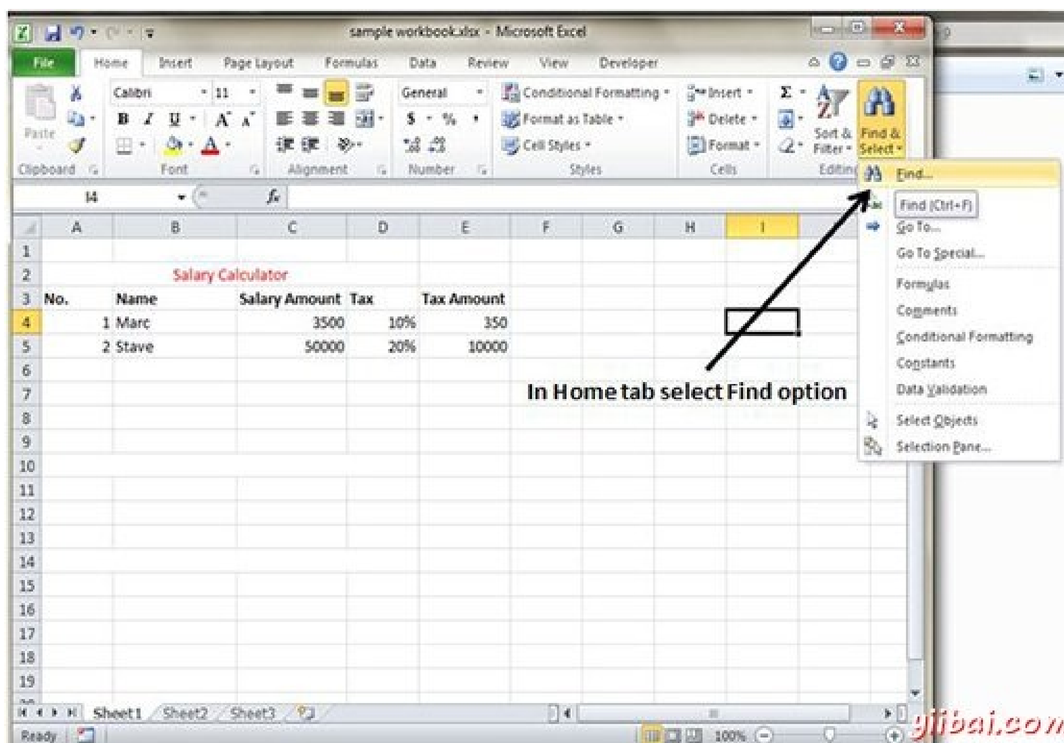
Excel查找和替换 - Excel教程

MS Excel提供了查找和替换在工作表中选项。

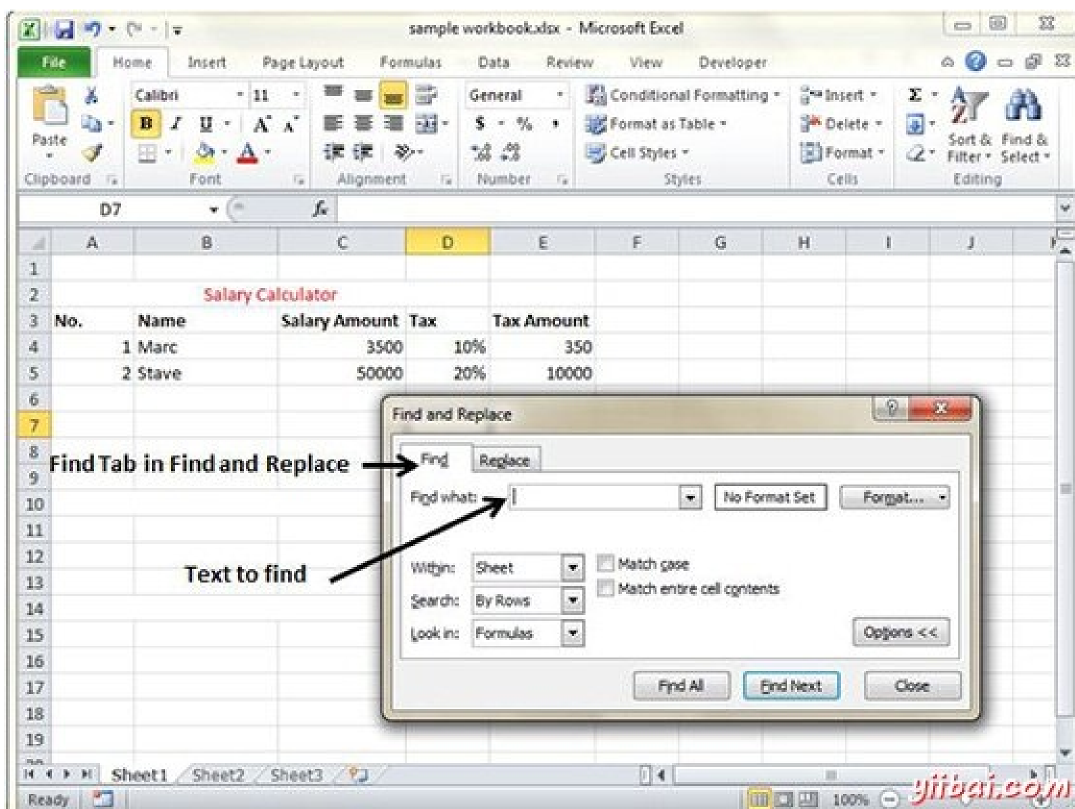
查找和替换对话

让我们来看看如何访问查找和替换对话。

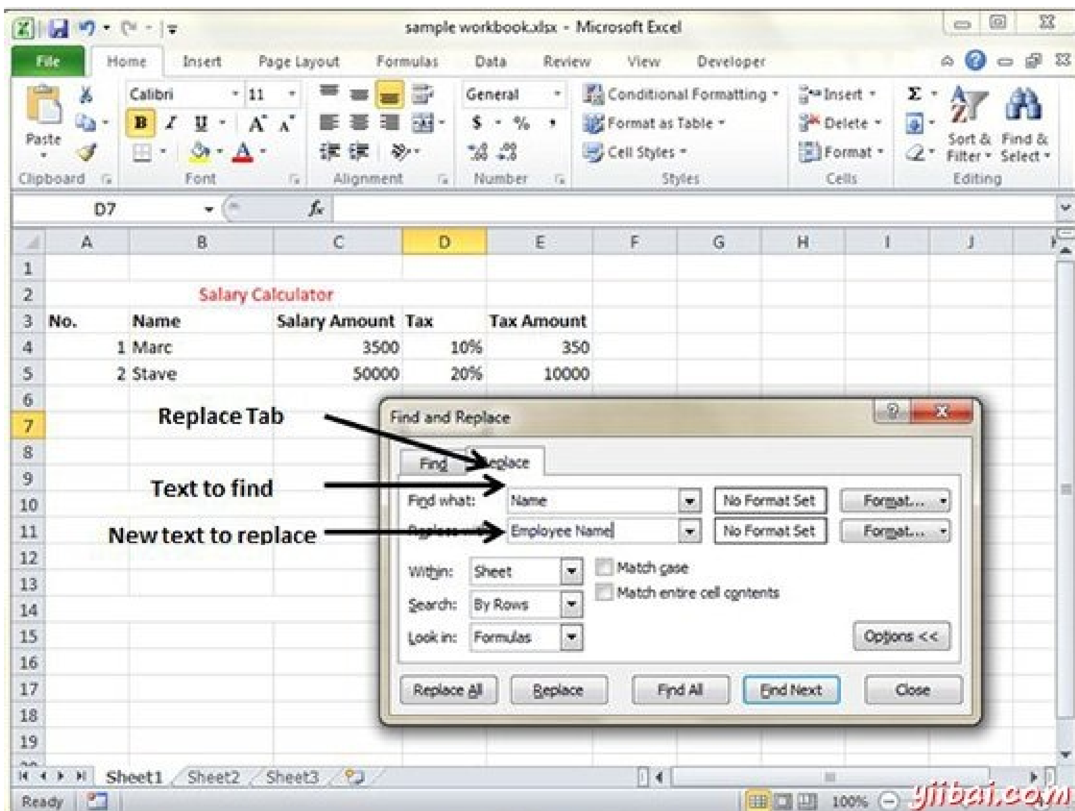
要访问查找和替换，选择 Home -> Find & Select -> Find 或按 Control + F Key如下文看到图片。



你可以看到查找和替换如下对话。



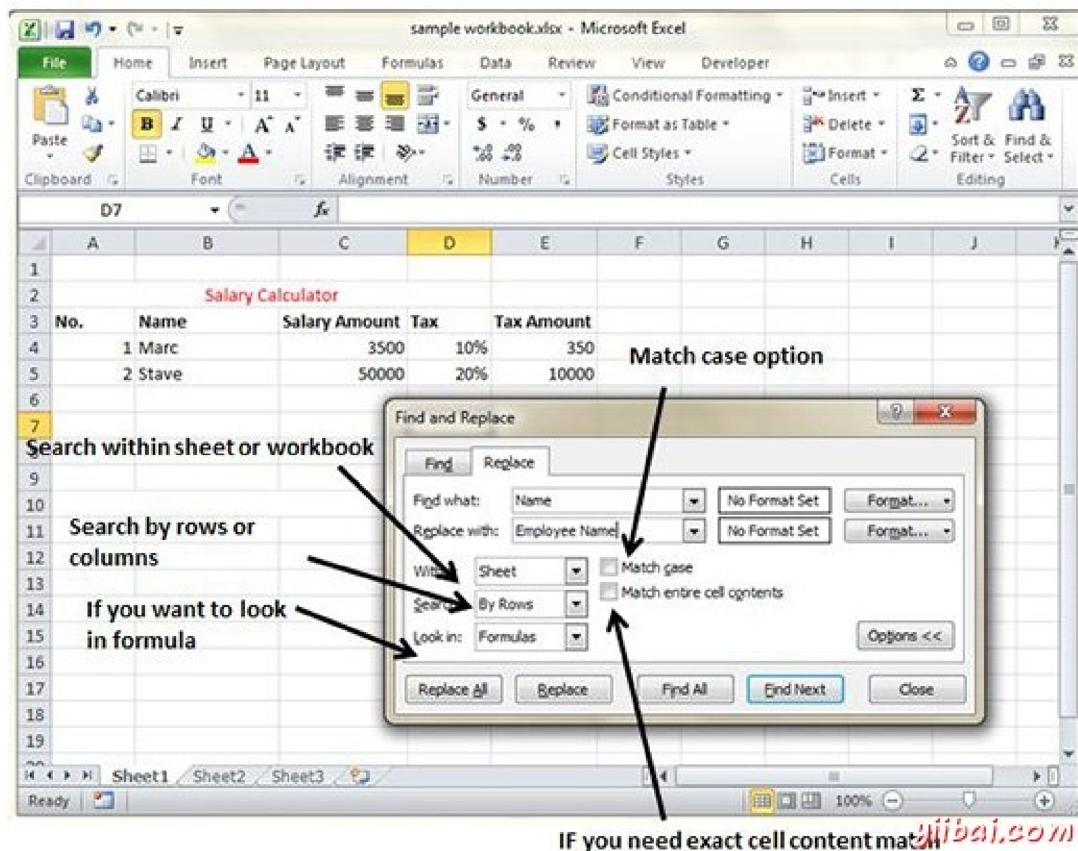
您可以替换为新的文本替换选项卡中找到的文本



浏览选项

现在，让我们来看看在查找对话框提供的各种选项。

- 内部：指定搜索应该是在工作表或工作簿。
- 按搜索：按行或列指定的内部搜索方法。
- 内查找：如果你想找到公式文本以及再选择此选项。
- 区分大小写：如果你想匹配类似小写字母或单词的大写的话，那么使用选项
- 匹配整个单元格的内容：如果你想精确随着单元格的词匹配，则选中此选项。



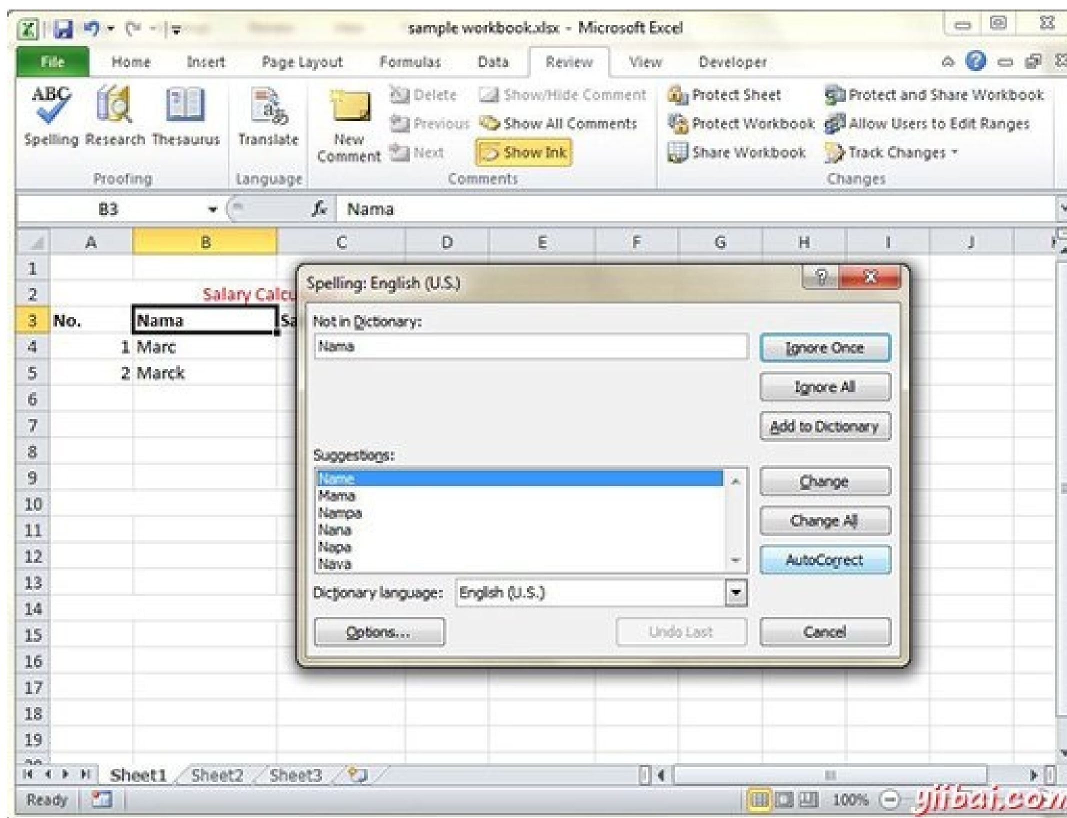
Excel拼写检查 - Excel教程

MS Excel提供的文字处理程序拼写检查功能。我们使用拼写检查功能的帮助可以摆脱拼写错误。

拼写检查的基础

让我们来看看如何访问拼写检查。

- 要访问拼写检查器，选择 Review ➪ Spelling 或者按 F7.
- 要检查特定范围的拼写，只需选择范围您在激活拼写检查之前。
- 如果拼写检查发现的任何字它不承认正确的，它会显示有建议选择拼写对话。



浏览选项

让我们来看看以拼写检查对话提供各种选项。

- 忽略一次：忽略词并继续拼写检查。
- 全部忽略：忽略这个词和它的所有后续出现。

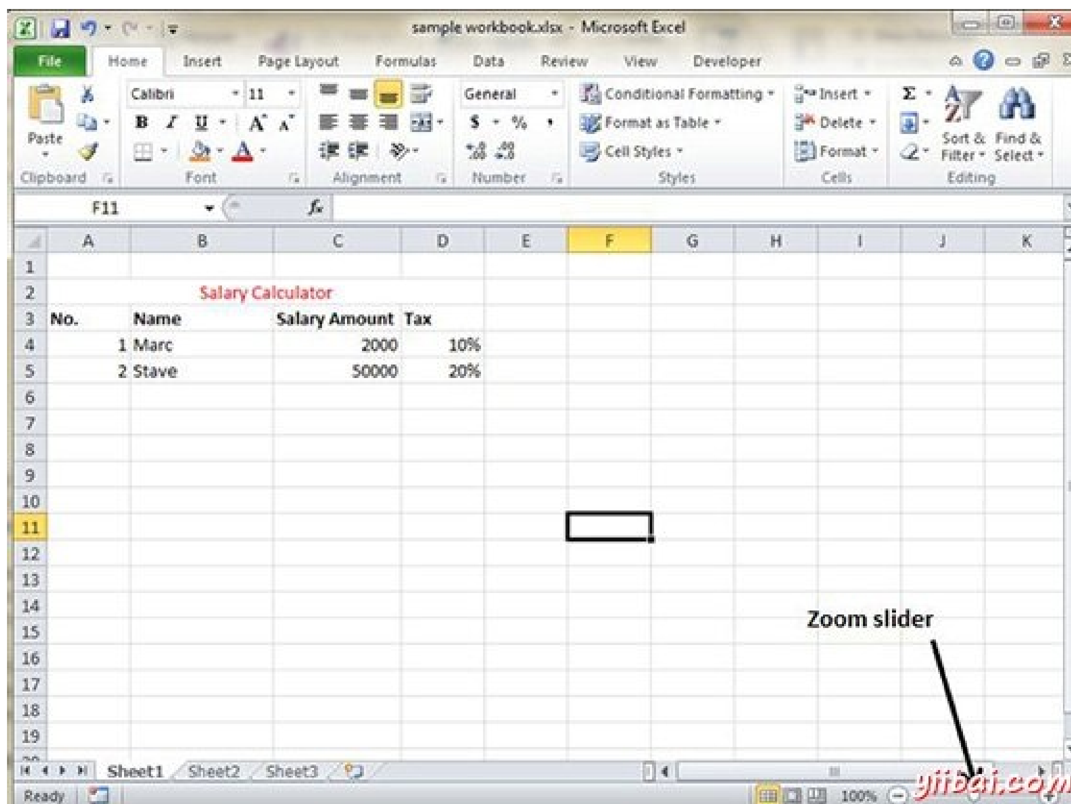
- 加入词典：单词添加到字典中。
- 更改：字更改为在建议列表中选择词。
- 全部更改：字更改为在建议列表中选择的单词，改变了这一切后续出现不再询问。
- 更正：拼错的单词和正确的拼写(你从列表中选择)添加到自动更正列表。

Excel放大/缩小 - Excel教程

缩放滑块

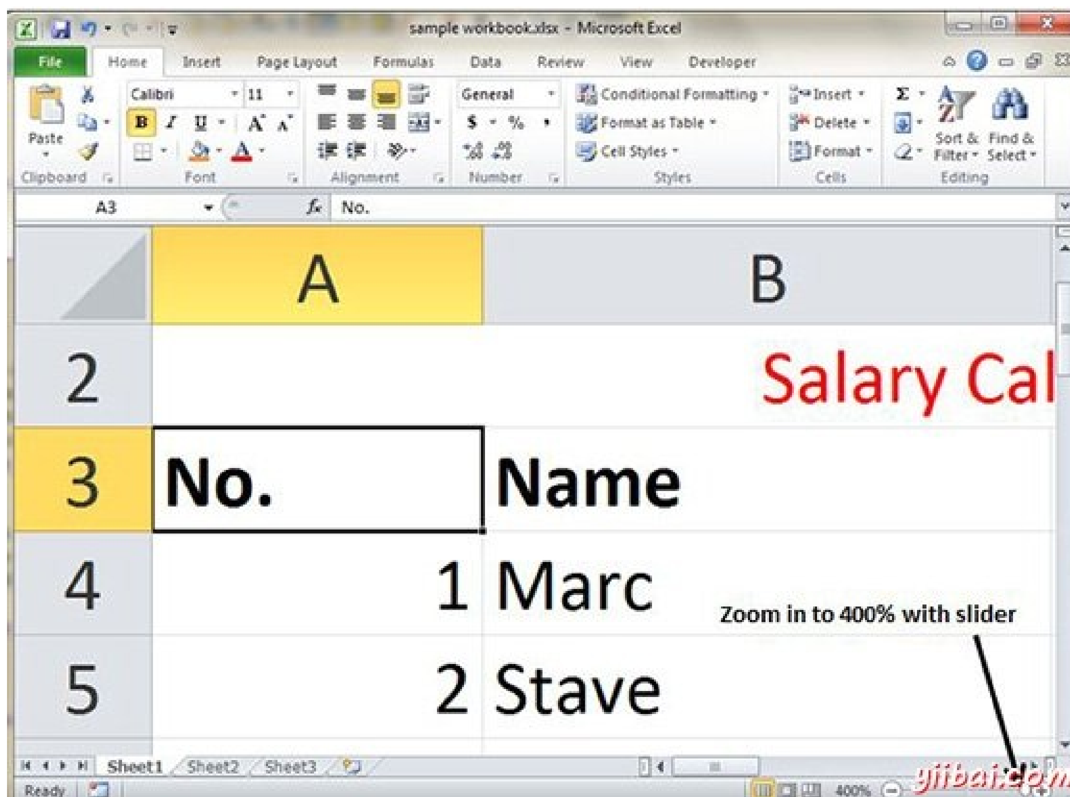
在MS Excel 屏幕上默认显示为100%。可以改变从10%(微小)，缩放比例为400%(巨大)。缩放不会改变的字体大小，因此它打印输出没有影响。

您可以在如下工作簿的右下角查看缩放滑块。



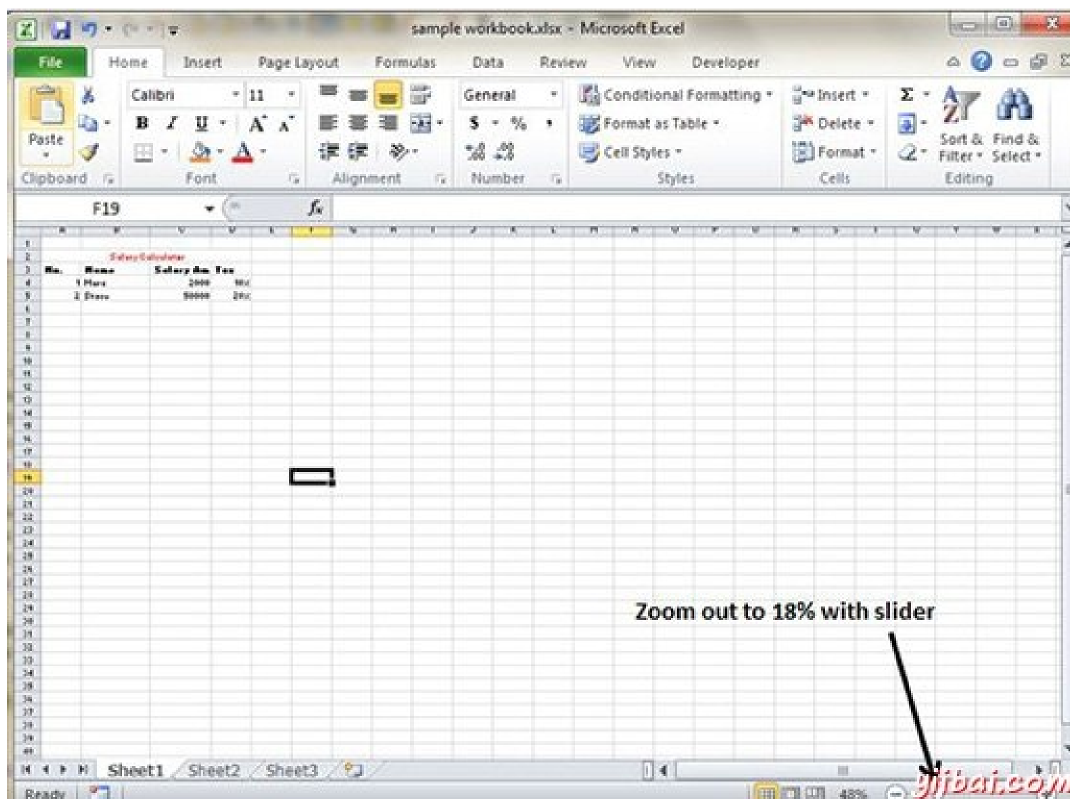
放大

您可以通过移动滑块向右放大工作簿。它将改变工作簿的只读视图。可以以最大400%缩放。请参阅下面的屏幕截图。



缩小

您可以通过移动滑块向左缩小工作簿。它将改变工作簿的只读视图。最大以10%缩放。参见下面的屏幕截图。



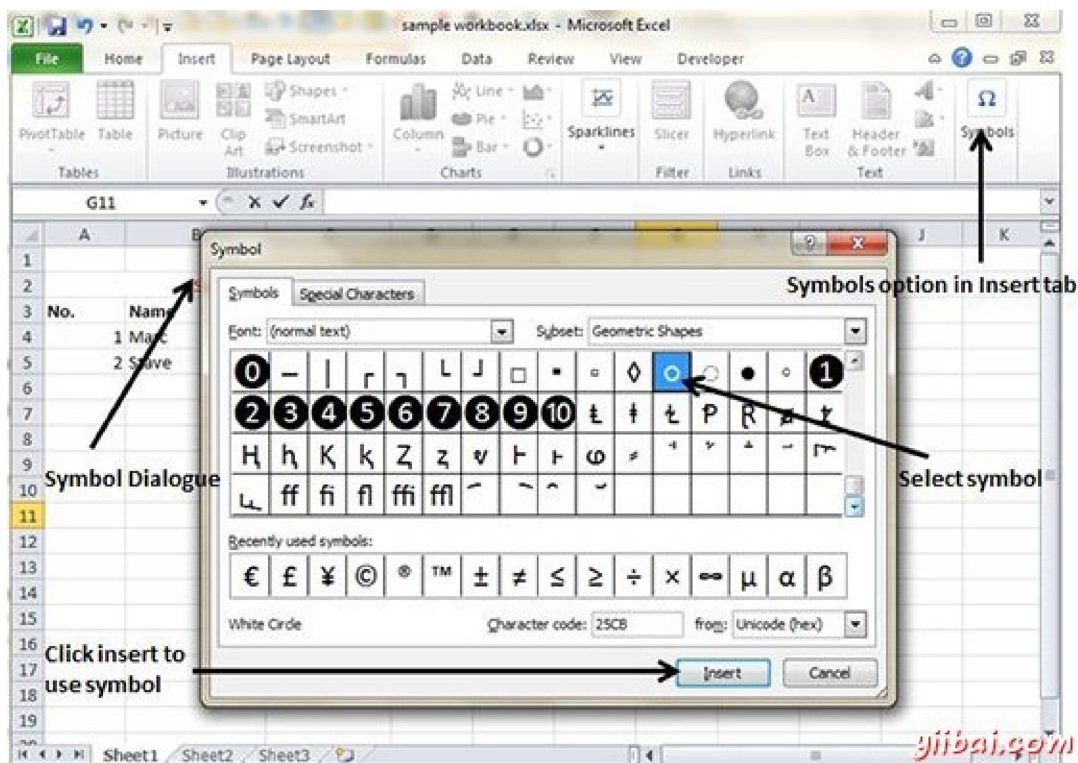
Excel特殊符号 - Excel教程

在某些情况下如果你想插入一些符号或特殊字符未在键盘上找到，使用的符号选项。

使用符号

转到 插入»符号»特殊字符 查看可用的符号。你可以看到许多可用的符号有像 Pi，α，β等

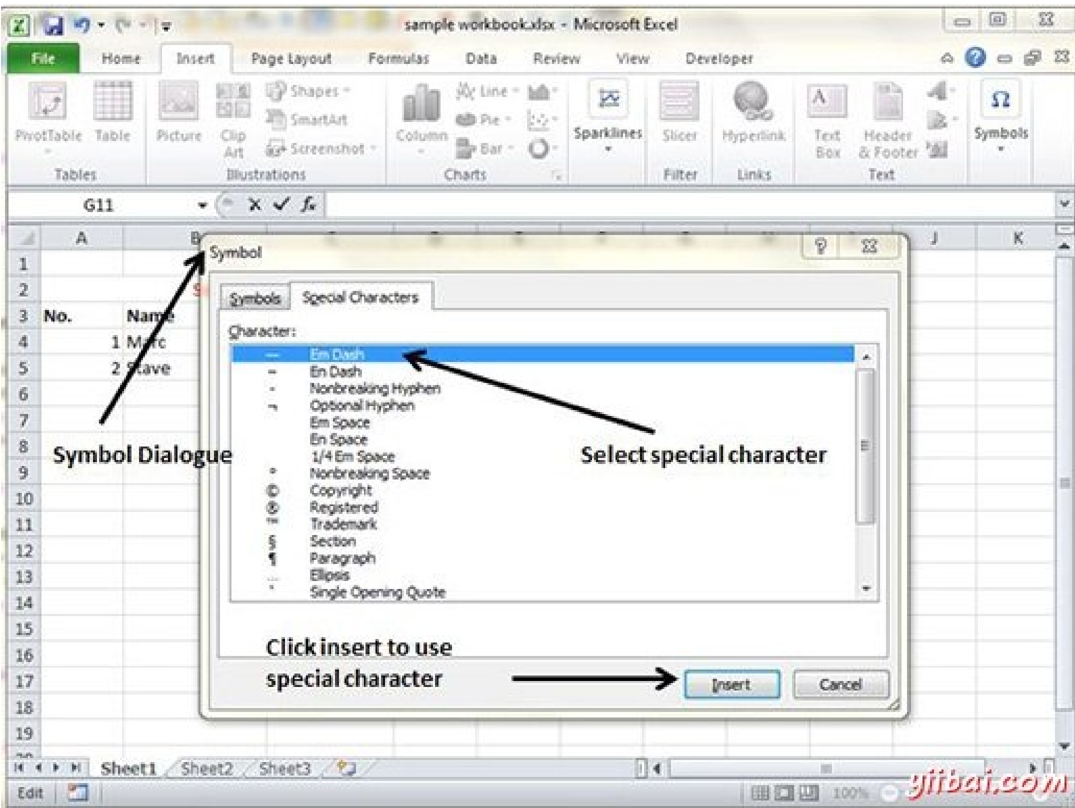
选择要添加，然后单击插入使用该符号



使用特殊字符

转到插入»符号»特殊字符，查看可用的特殊字符。你可以看到许多特殊字符，可有类似版权，注册等

选择要添加，然后单击插入使用特殊字符的特殊字符



sample workbook.xlsx - Microsoft Excel
Symbol
Symbols
Special Characters
Character:
Em Dash
En Dash
Nonbreaking Hyphen
Optional Hyphen
Em Space
En Space
1/4 Em Space
Nonbreaking Space
Copyright
Registered
Trademark
Section
Paragraph
Ellipsis
Single Opening Quote
Symbol Dialogue
Select special character
Click insert to use
special character
Insert
Cancel

Excel插入注释 - Excel教程

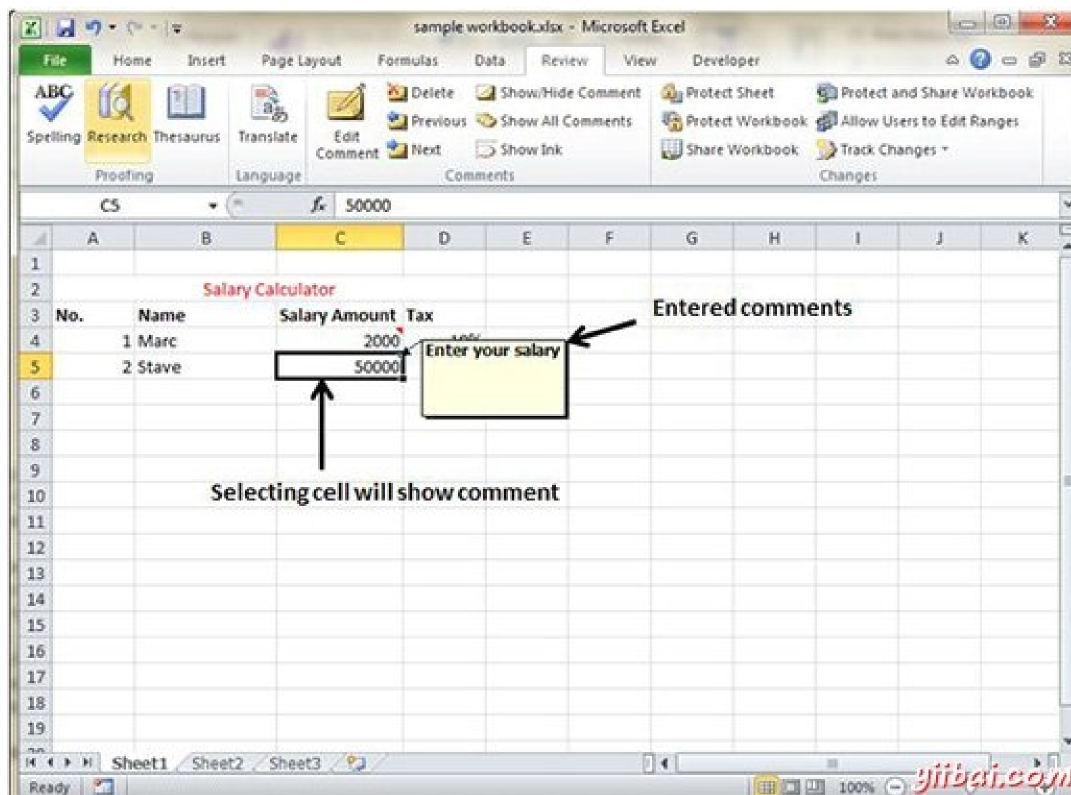
添加批注到单元格

加入到单元格注释有助于理解单元格的目的，它应该有什么输入，等等。它有助于理解文档

若要添加注释单元格选择单元格，执行下面的动作

- 选择审查 » 注释 » 新注释
- 右键单击该单元格，然后从可用选项插入注释。
- 按 Shift+F2

最初发表注释包括计算机的用户名。你可对文本的单元格批注进行修改

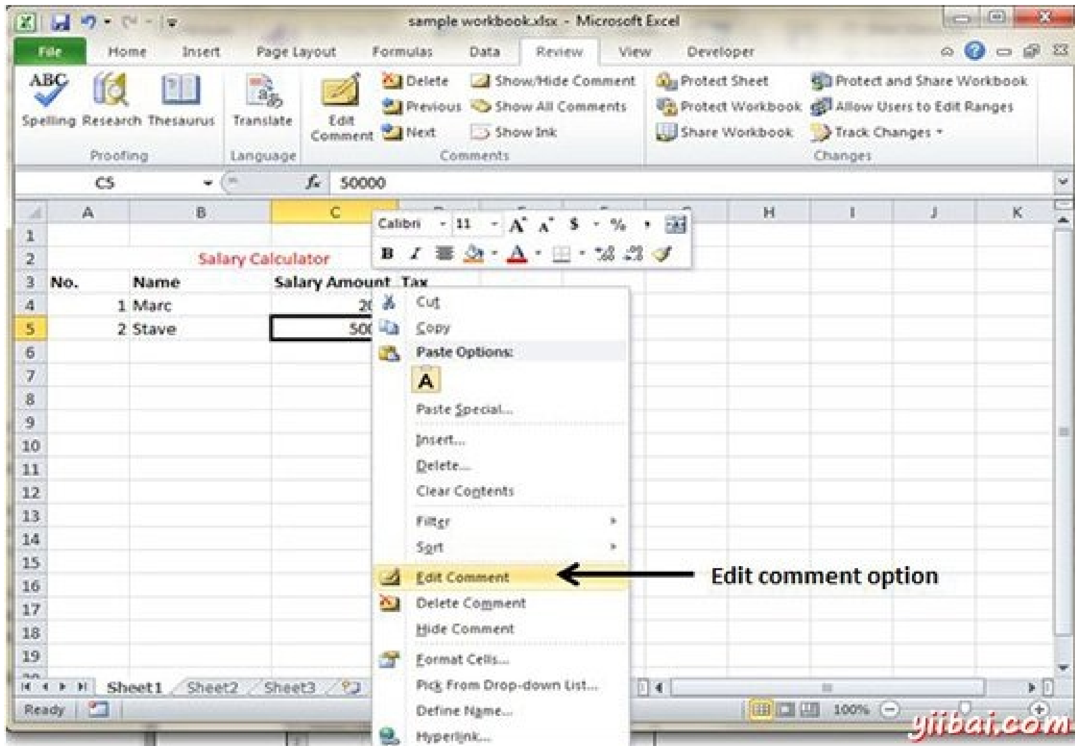


修改注释

您可以修改您之前进入注释如下。

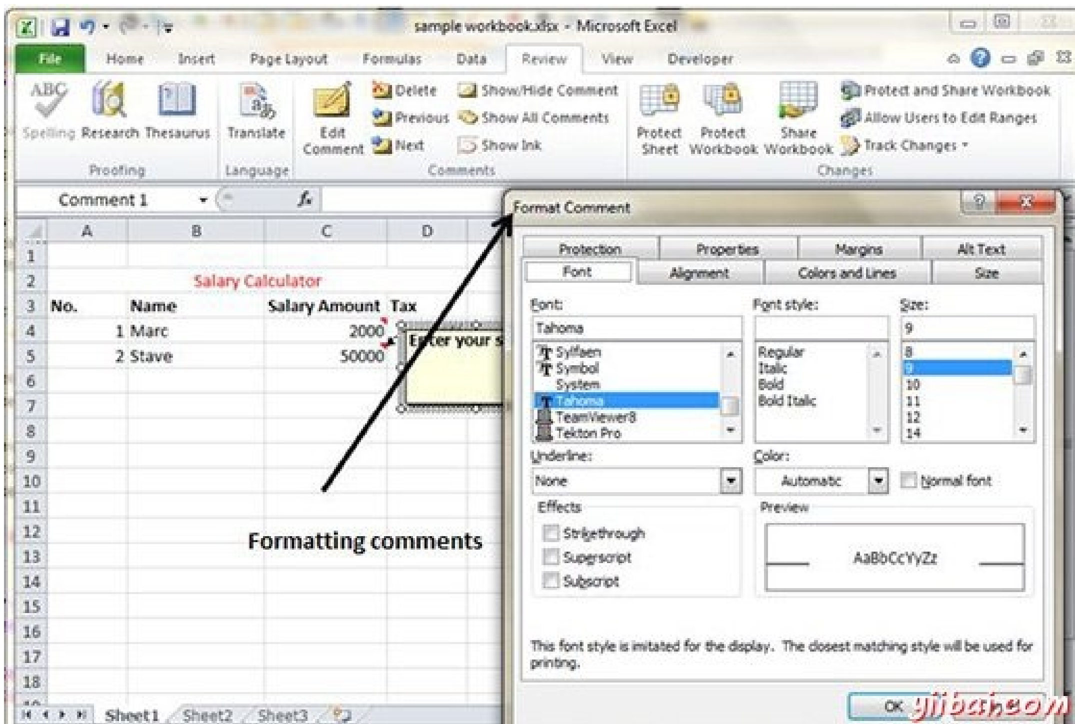
- 选择要在其注释出现的单元格。
- 右键单击该单元格，然后从可用选项编辑注释。

- 修改注释



格式化注释

有几种格式化注释。对于格式化注释右键单击单元格»编辑注释»选择注释»右击它»格式发布注释，你可以改变颜色，字体，大小等方面来格式化注释。



Excel添加文本框 - Excel教程

文本框

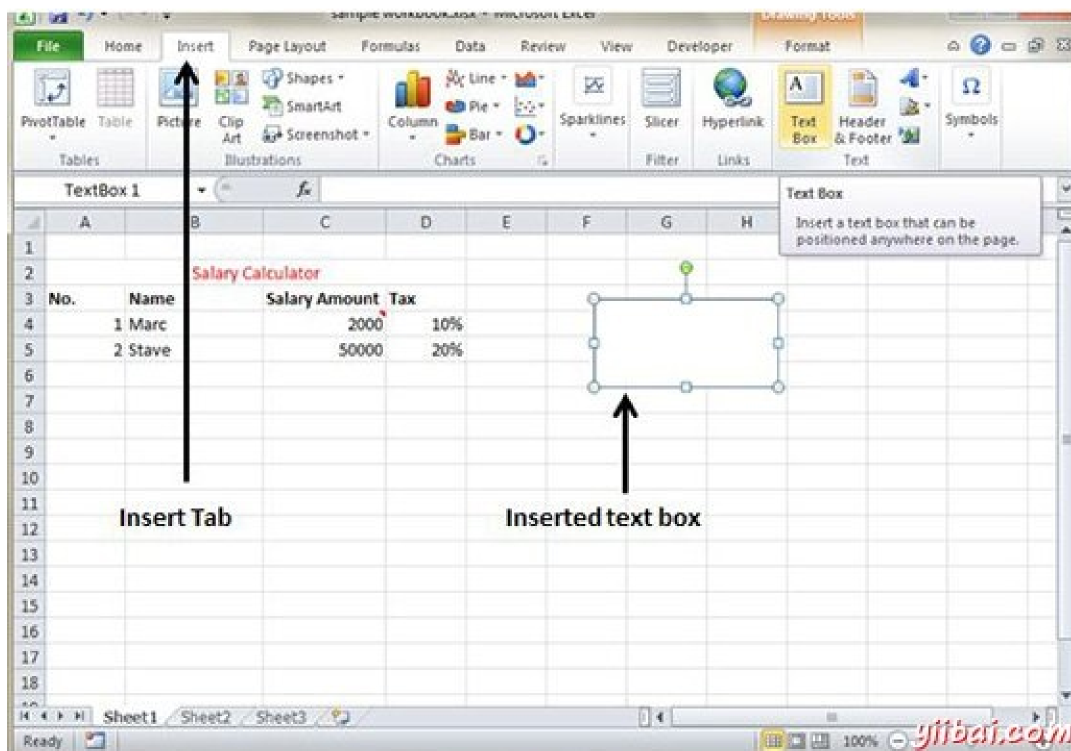
文本框是结合文字有矩形图形对象的特殊图形对象。文本框和单元格注释是，他们显示相似矩形框中的文本，但文本框始终可见，而单元格批注成为选择后单元格可见。

添加文本框

要添加操作文本框，如下执行。

- 选择插入»文本框»选择文本框或绘制

最初发表注释包括计算机的用户名。你可对文本的单元格批注进行修改

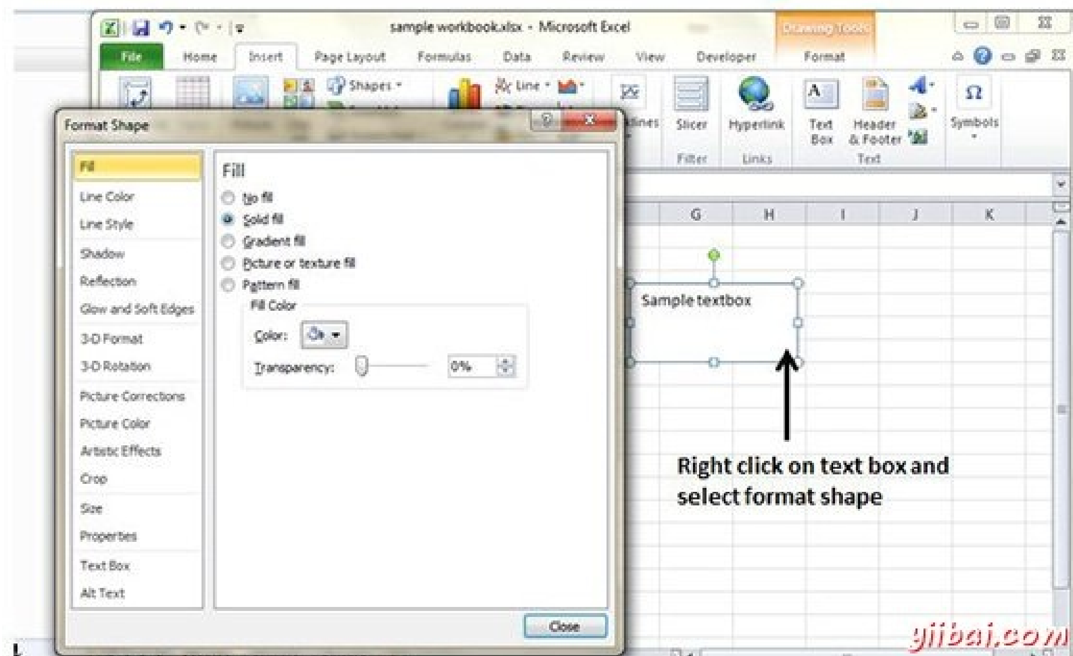


格式文本框

当你添加了文本框，您可以通过更改字体，字体大小，字体样式和对齐方式等格式化文本框。让我们来看看一些重要的选项那里。

- 填充：指定填充的文本框一样，没有填充，实心填充。同时指定的文本框填充透明度。
- 行颜色：指定线颜色的线和透明度。

- 行样式：指定行样式和宽度。
- 大小：指定文本框的大小。
- 属性：指定文本框的一些属性。
- 文本框：指定文本框的布局，自动调整选项和内部空间。

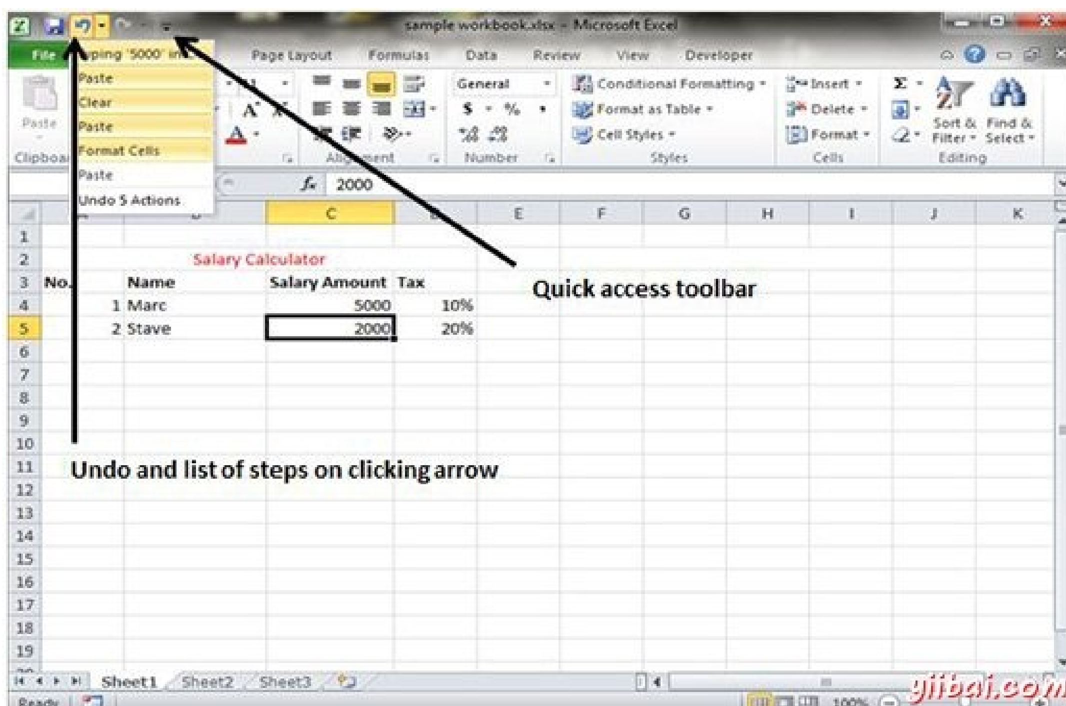


Excel撤消更改 - Excel教程

撤消更改

您可以在Excel中通过使用撤消命令，几乎可撤消每一个动作。我们可以使用2种方式撤消更改。

- 从快速访问工具栏»点击撤消
- 请按Ctrl+ Z

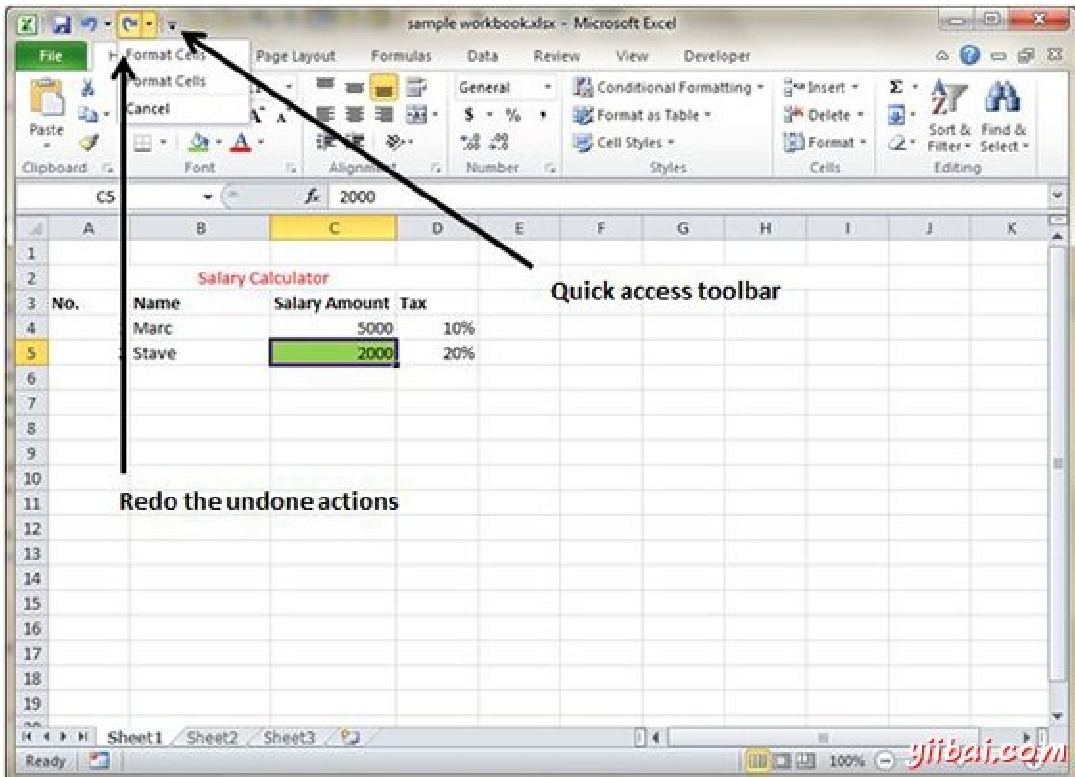


可以反向通过执行撤消不止一次执行在过去的100动作的效果。如果你点击右侧的撤销按钮的箭头，你可以扭转操作的列表。单击某个项目在该列表撤消行动，所有你执行的后续操作。

重做更改

可以再次转回并撤消在Excel中执行的操作，使用重做命令。我们可以重做改变2种方式。

- 从快速访问工具栏»点击重做
- 请按Ctrl+ Y

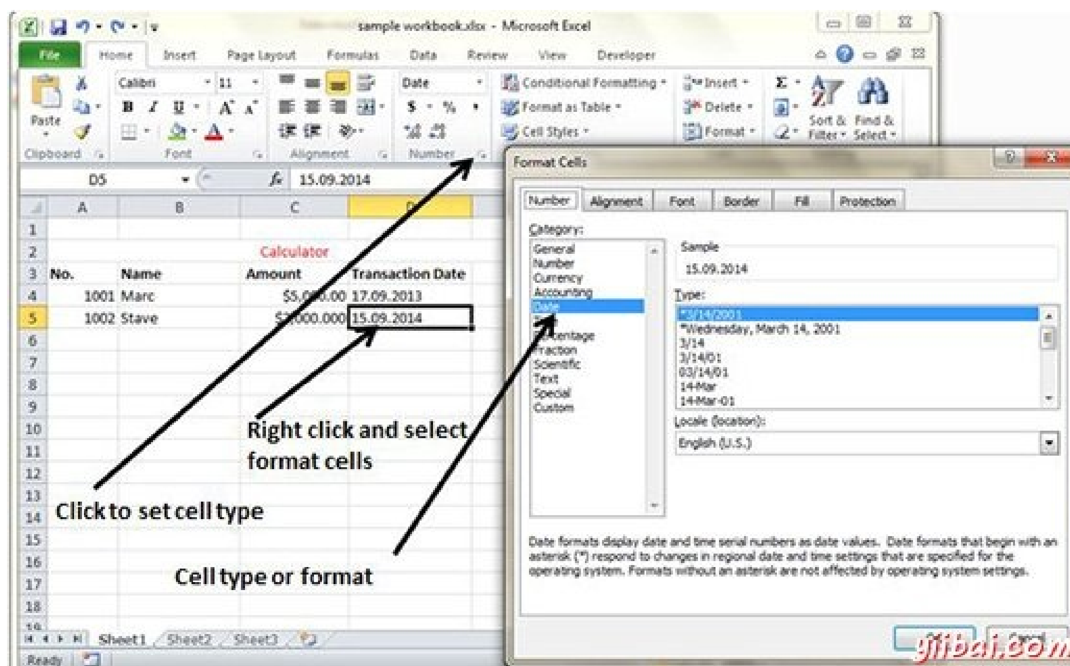

sample workbook.xlsx - Microsoft Excel
Quick access toolbar
Redo the undone actions
Salary Calculator
No.
Name
Salary Amount
Tax
Marc
5000
10%
Stave
2000
20%

Excel设置单元格类型 - Excel教程

格式化单元格

MS Excel单元格可以容纳不同类型，如数字，货币，日期等数据..您可以以各种方式设置下面的单元格类型：

- 右键单击单元格»设置单元格格式»数值
- 点击色带上的色带



各种单元格的格式

以下是各种单元格的格式。

- 通用：这是单元格的默认单元格的格式。
- 数值：这将显示单元格与分隔数
- 货币：该单元格显示货币即带有货币符号。
- 会计：类似于货币用于会计目的。
- 日期：多种日期格式，这个类似17-09-2013，17th-Sep-2013等
- 时间：不同的时间格式，如1.30PM，13.30等
- 百分比：这显示单元格以及小数，如50.00%的百分比

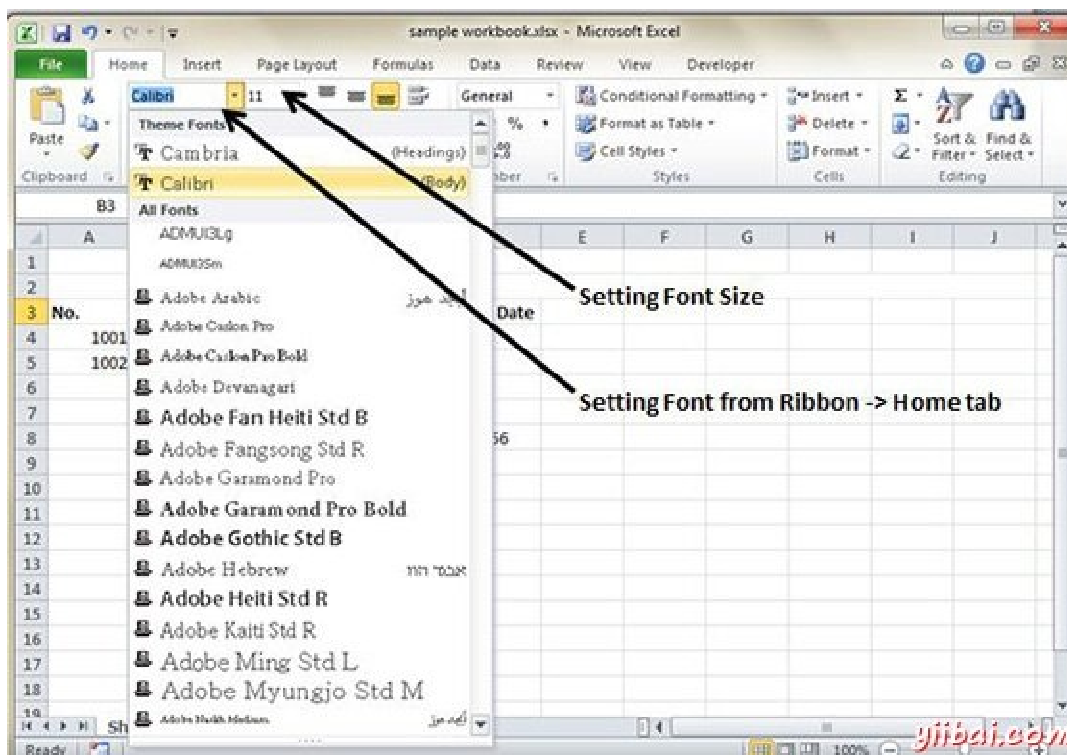
- 分数：显示单元格作为部分，如1/4，1/2等等
- 科学计数：显示单元格作为指数如 5.6E+01
- 文本：此单元格显示为正常文本。
- 特别声明：这是一个特殊的格式，单元格像邮编，电话号码
- 自定义：可以通过使用自定义格式。

Excel设置字体 - Excel教程

可以将任意已安装的单元格在工作表字体到打印机。

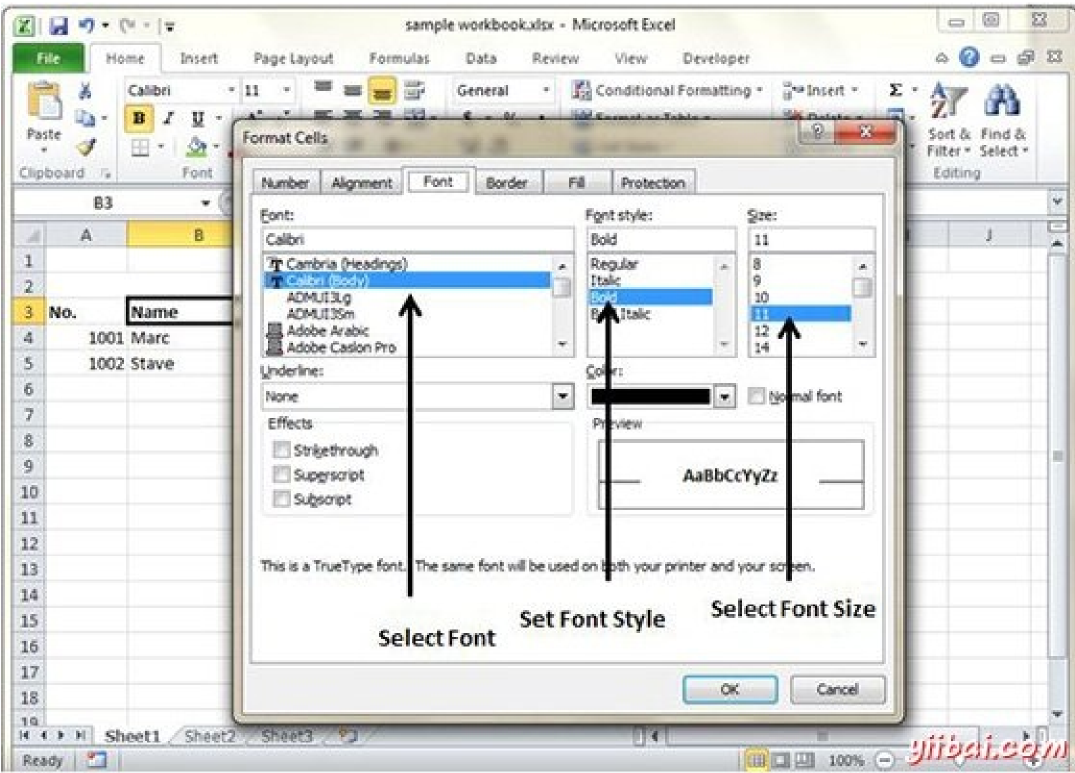
从主页设置字体

您可以从设置所选文本字体 Home » Font group » 选择字体



从格式单元格对话设置字体

- 右键点击 单元格 » 设置单元格格式 » 字体选项
- 按 Control + 1 或 Shift + Control + F


sample workbook.xlsx - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Format Cells
Number
Alignment
Font
Border
Fill
Protection
Font:
Font style:
Size:
Calibri
Bold
11
Underline:
None
Effects
Strikethrough
Superscript
Subscript
Preview
AaBbCcYyZz
This is a TrueType font. The same font will be used on both your printer and your screen.
Select Font
Set Font Style
Select Font Size
OK
Cancel
No.
Name
1001 Marc
1002 Stave
Sheet1
Sheet2
Sheet3
Ready
yiibai.com

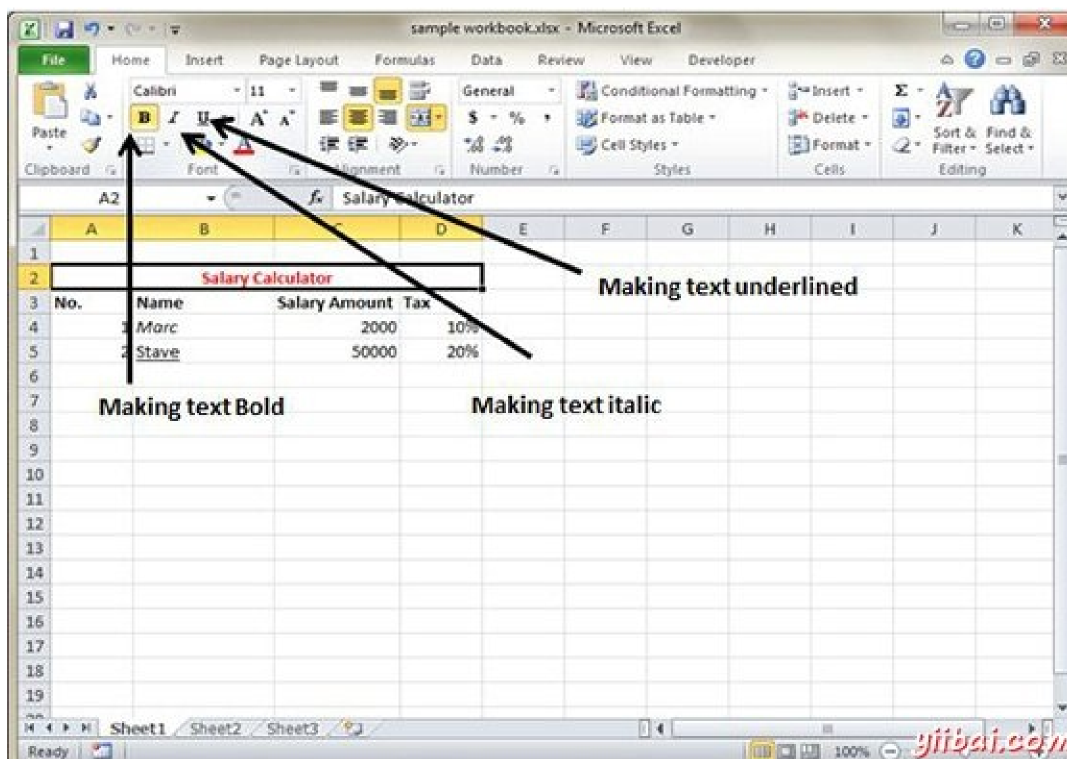
Excel文字装饰 - Excel教程

可以更改单元格的文本装饰来改变它的外观和感觉。

文字修饰

在如下主页功能区的选项卡中提供的各种选项

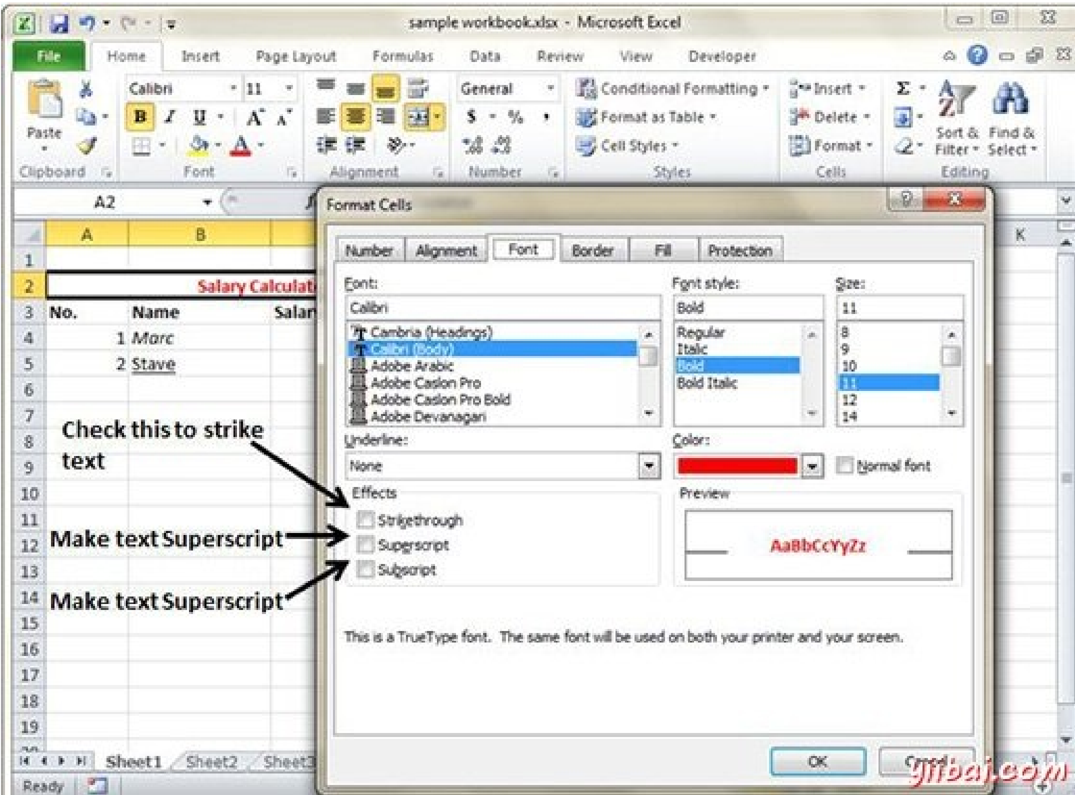
- Bold : 它使文本由选择首页»字体组»点击B或请按Ctrl+ B凸显
- Italic : 它使文本倾斜通过选择首页»字体组»点击I 或请按Ctrl+ B
- Underline : 它使文本通过选择首页»字体组»点击U或请按Ctrl+ B下划线
- Double Underline : 它使文本强调选择首页»字体组»点击箭头附近U»选择双下划线



更多的文字装饰选项

可在下面的文字装饰更多选择 格式化单元»字体标签»影响单元格

- 删除线：这使得删除线在文本中心垂直
- 超级脚本：它使内容显示为超
- 子脚本：它使内容显示为子

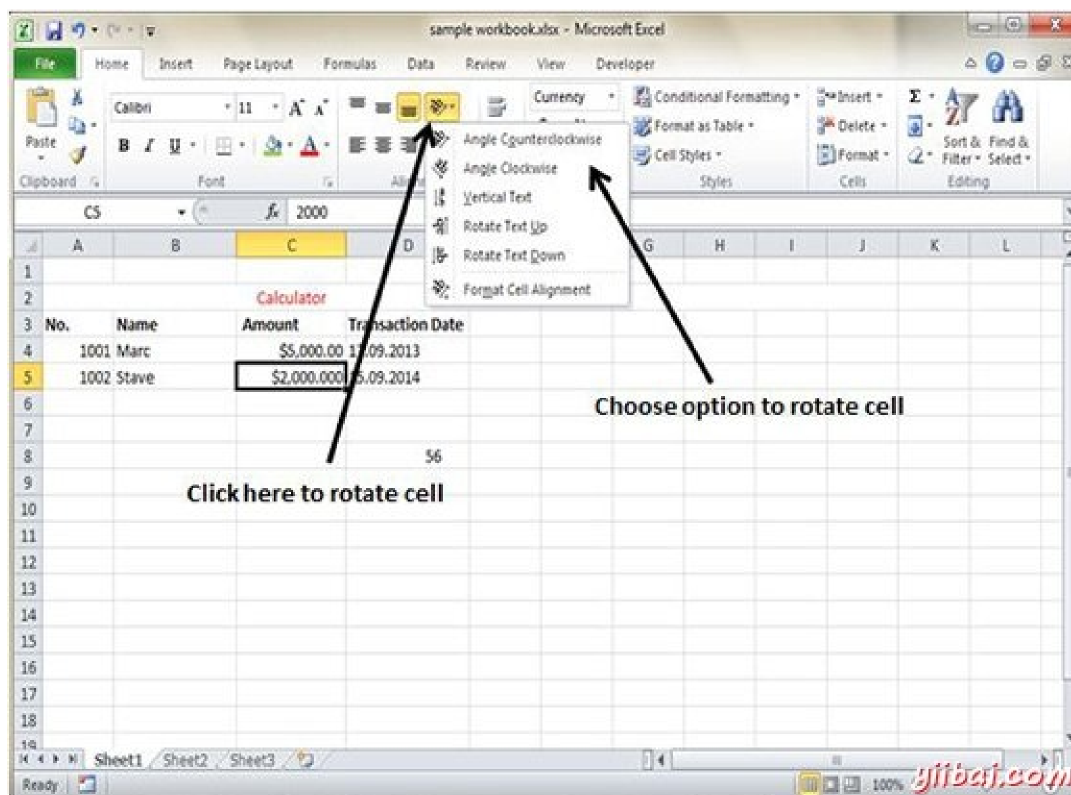

sample workbook.xlsx - Microsoft Excel
Format Cells
Number
Alignment
Font
Border
Fill
Protection
Font:
Font style:
Size:
Underline:
Color:
Normal font
Effects
Strikethrough
Superscript
Subscript
Preview
AaBbCcYyZz
This is a TrueType font. The same font will be used on both your printer and your screen.
OK
Check this to strike text
Make text Superscript
Make text Superscript
Salary Calculati
No.
Name
1 Marc
2 Stave
Sheet1
Sheet2
Ready
yiibai.com

Excel旋转单元格 - Excel教程

可以通过任何度旋转单元格以改变单元格的定向。

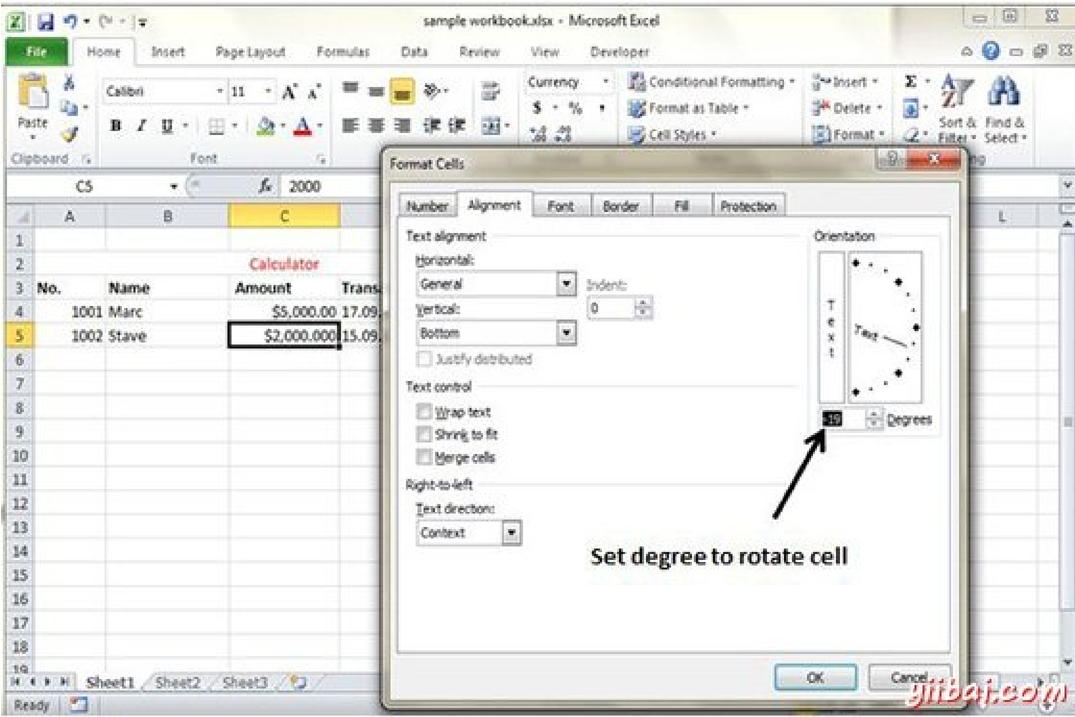
旋转格从主页选项卡

点击在主页选项卡的方向。选择如角度逆时针，顺时针角度等可供选择



从格式化单元格旋转格

右键单击该单元格。选择设置单元格格式»对齐»设置度旋转

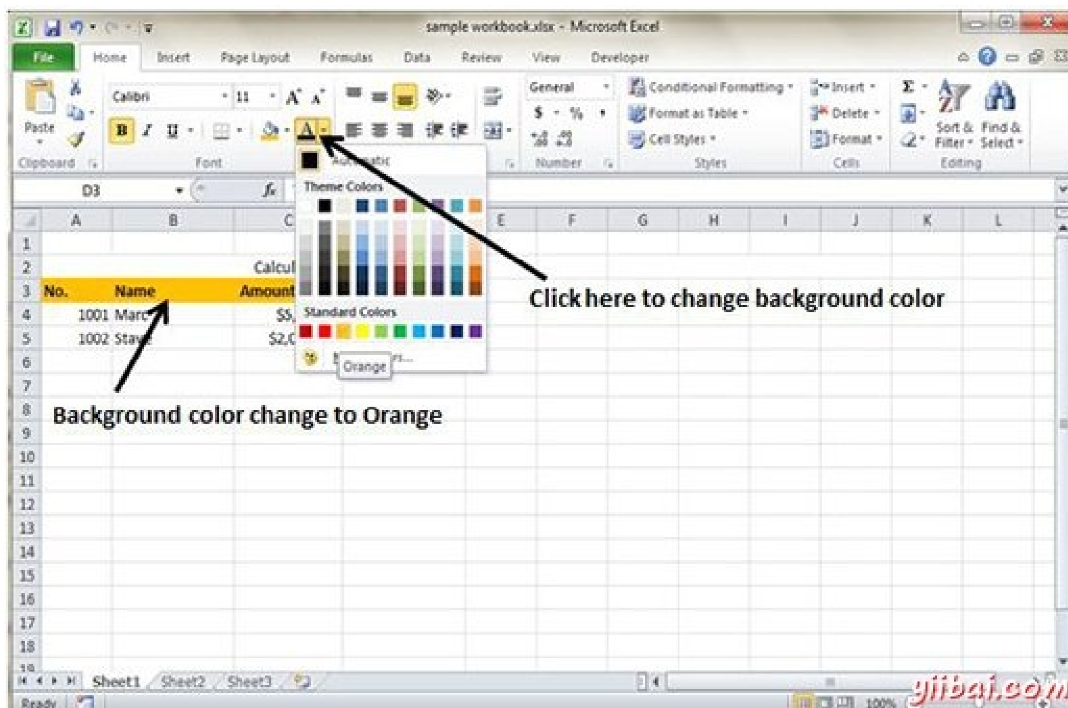

Set degree to rotate cell

Excel设置颜色 - Excel教程

可以更改单元格或文本颜色的背景颜色。

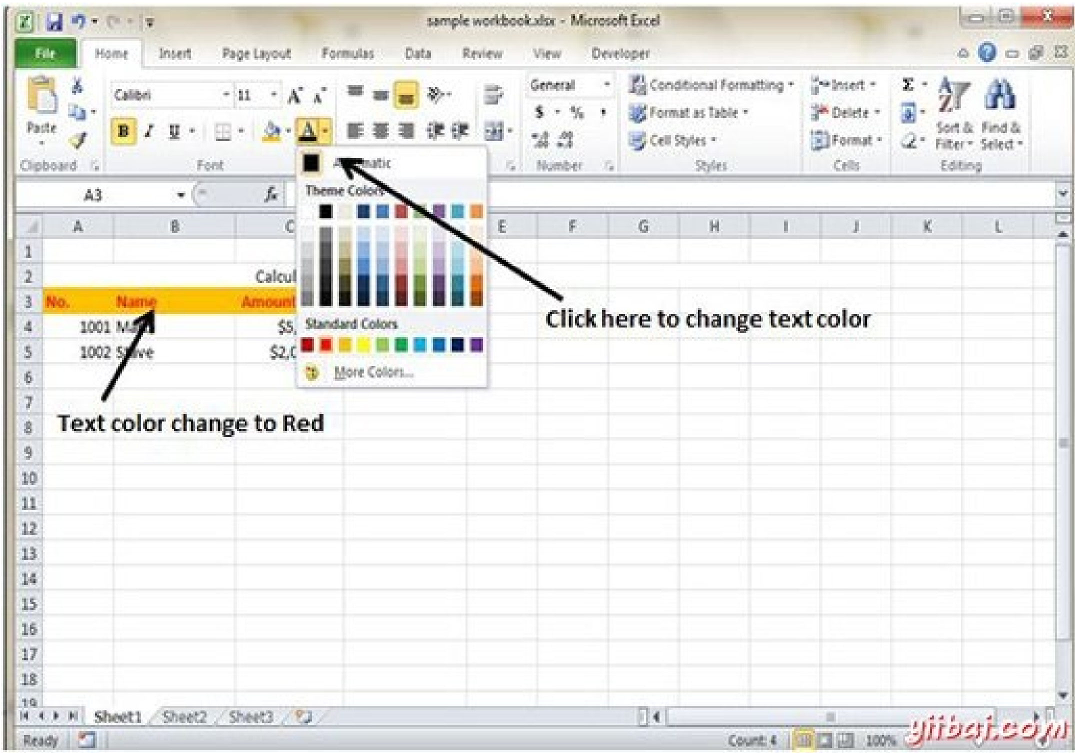
更改背景颜色

在MS Excel默认的单元格的背景颜色是白色。你可以把它按需要更改 首页标签»字体组»背景颜色

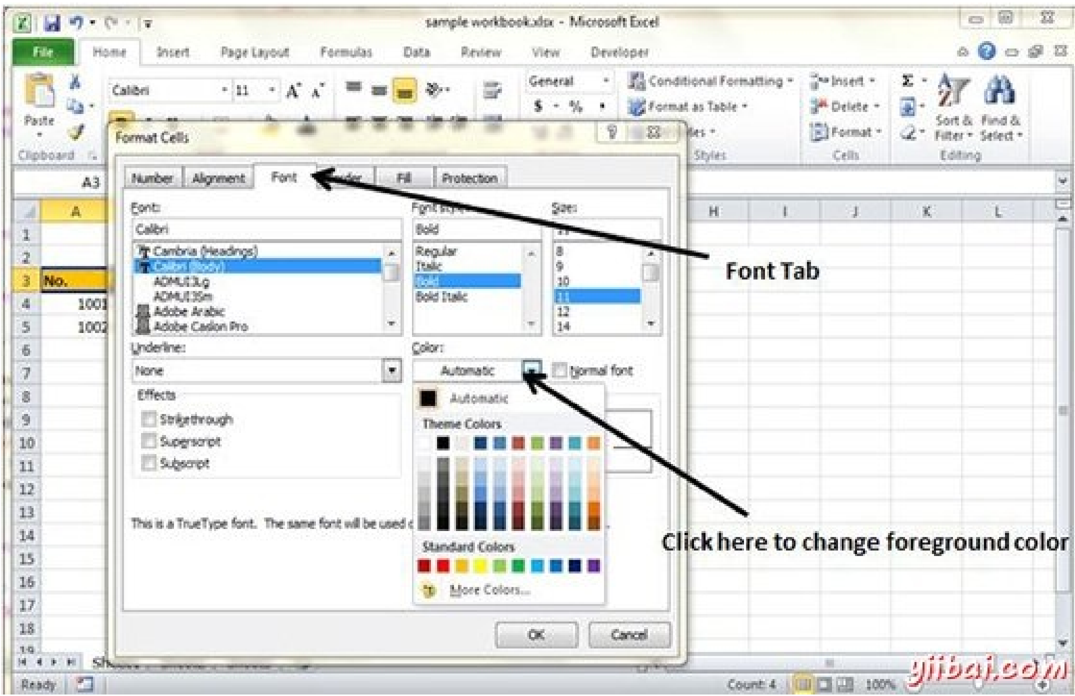


更改前景色

在MS Excel中，默认情况下，前景或文本颜色为黑色。你可以把它按您的需要更改 首页标签»字体组»前景色



也可以通过选择单元格更改前景色 右键单击»设置单元格格式»字体选项卡»颜色

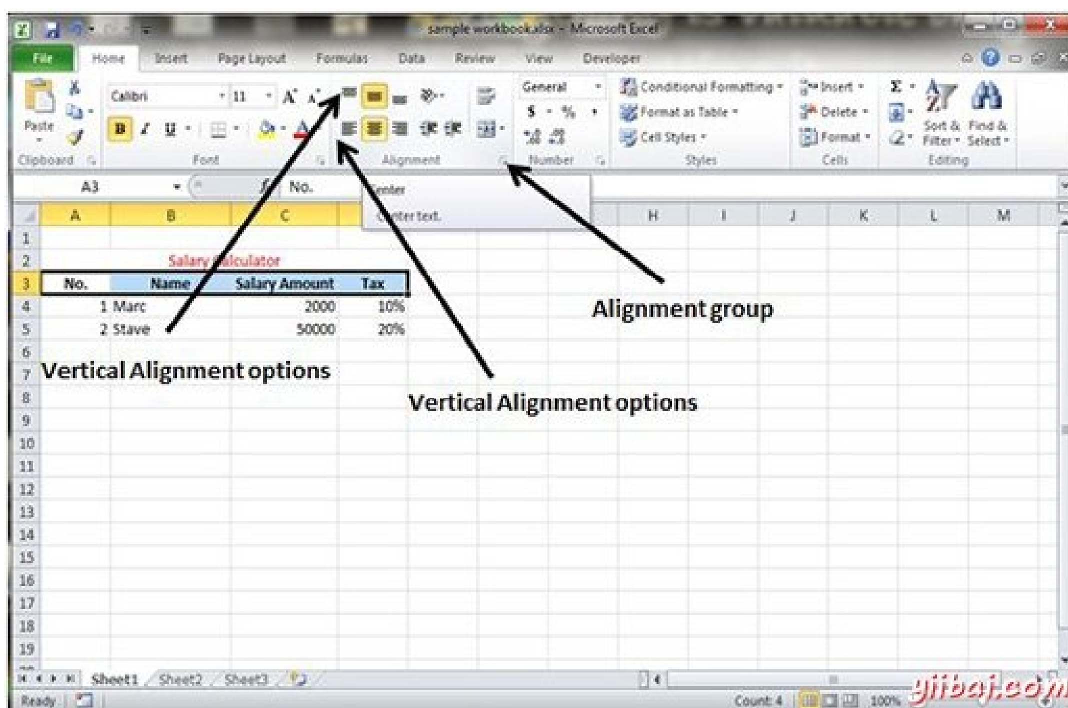


Excel文字对齐方式 - Excel教程

如果你不喜欢单元格的默认对齐方式，您可以更改单元格对齐方式。下面是更改单元格对齐方式的各种方法。

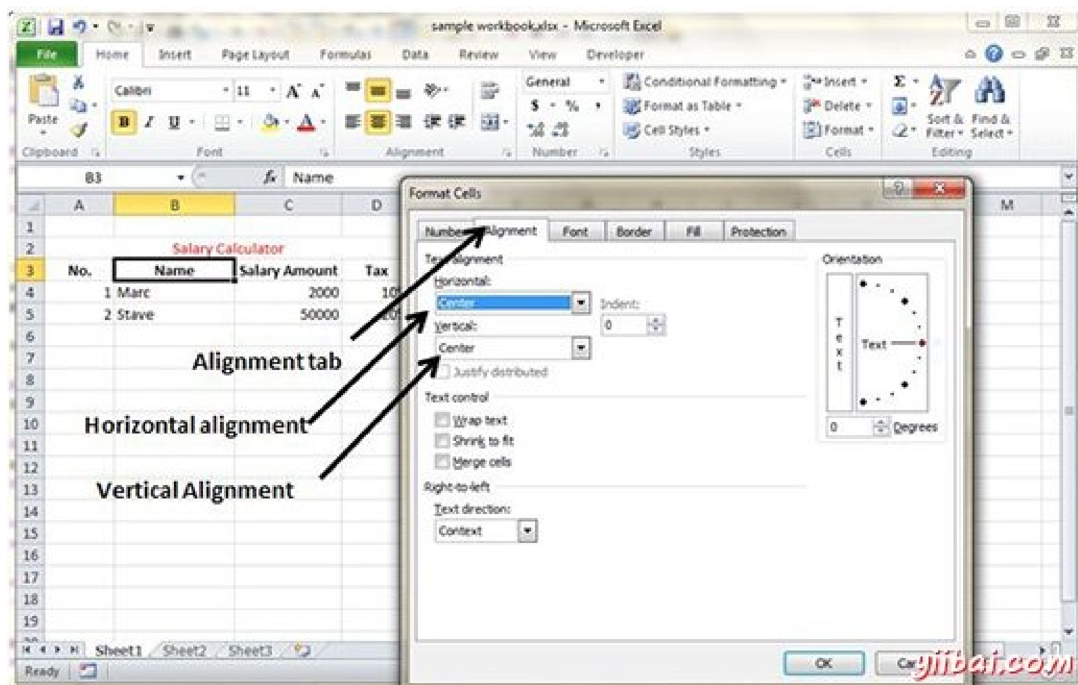
从主页选项卡中更改对齐方式

你可以更改单元格的水平和垂直对齐。默认Excel对齐数字向右，文本则是左边对齐。点击对齐方式组中的可用选项选项卡以更改对齐。



从设置单元格格式更改对齐方式

右键单击单元格，然后选择格式的单元格。在单元格格式对话框中选择对齐选项卡。选择从垂直对齐和水平对齐选项可用选项



浏览对齐选项

1. 水平对齐：可以设置水平对齐方式为左，中心，右等

- 左：对齐单元格内容到单元的左侧。
- 中心：居中单元格内容。
- 右：校准单元格内容向右侧的单元格。
- 填充：重复单元的内容，直到该单元格的宽度被充满。
- 两端对齐：对齐文本到小区的左边和右边对齐。此选项仅适用如果单元格的格式设置为文本包裹，并使用多行。

2.垂直对齐方式：您可以设置垂直对齐到顶部，中部，底部等等。

- 居顶部：校准单元格内容到单元的顶部。
- 居中：居中的单元格内容垂直在单元格。
- 居底部：校准单元格内容到单元的底部。
- 两端对齐：两端对齐竖直单元格中的文本; 此选项仅适用如果单元格的格式设置为文本包裹，并使用多行。

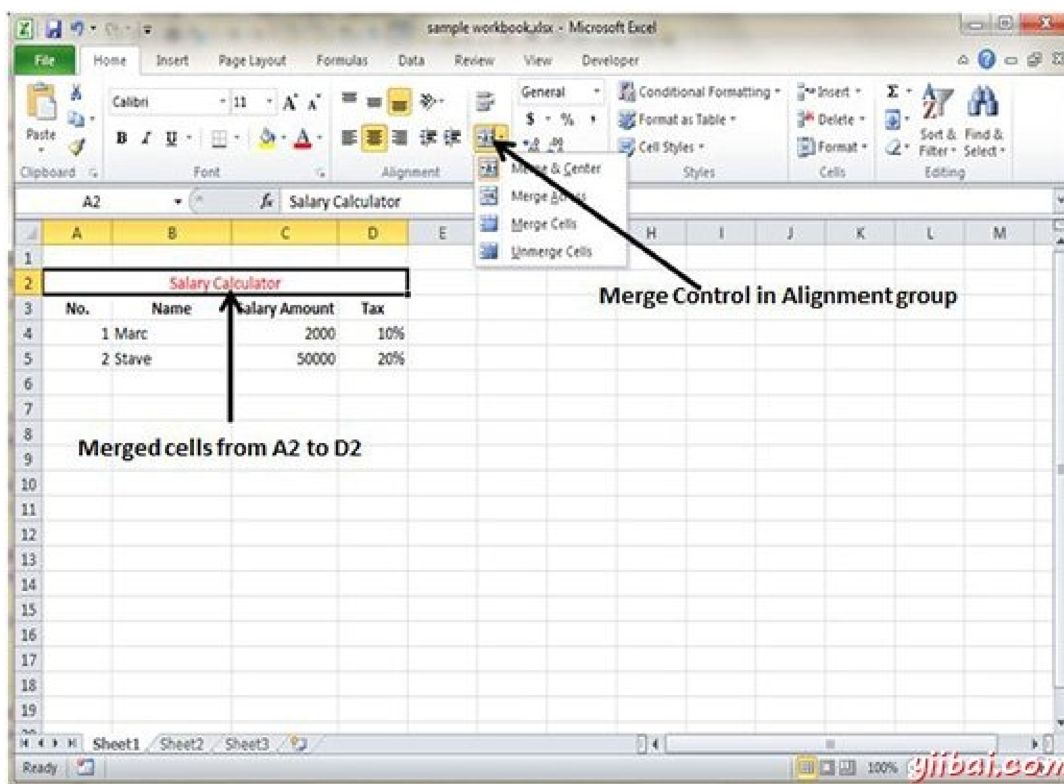
Excel合并及换行 - Excel教程

合并单元格

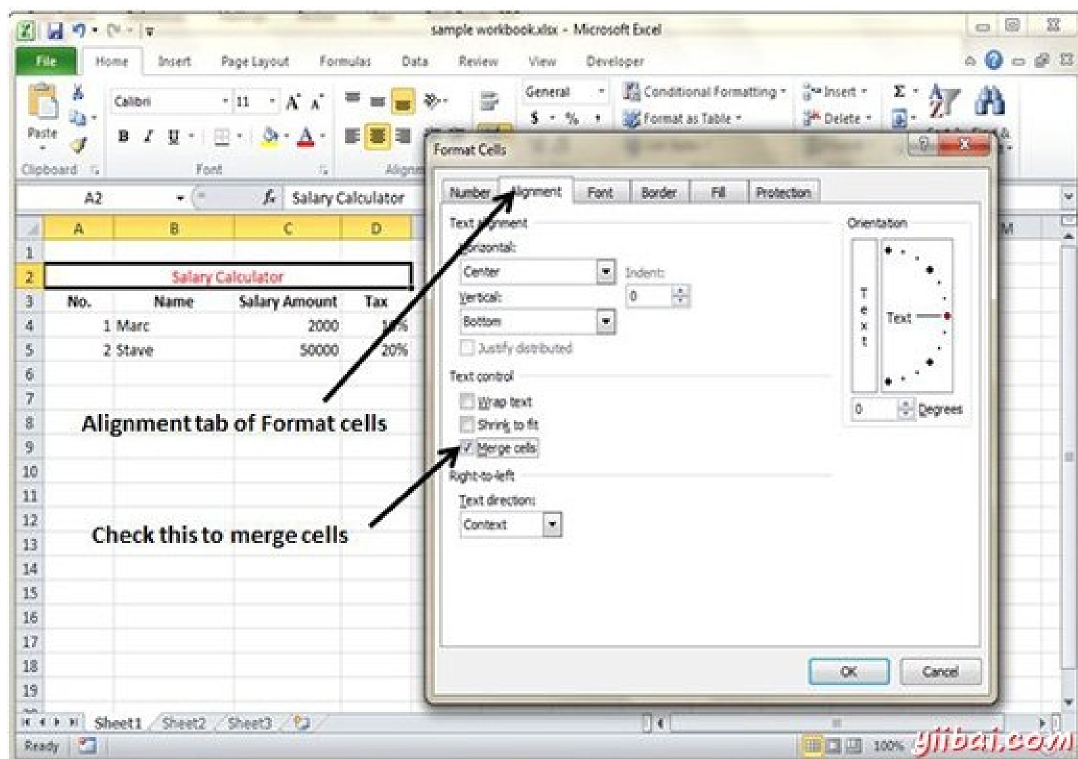
MS Excel中，您可以合并两个或多个单元格。当你合并单元格，您不要在单元格的内容。相反，结合一组单元格向占据同一空间中的单个单元格。

您可以通过如下多种方式合并单元格

- 选择合并与中心控制功能区上更简单。要合并单元格，选择想要合并的然后单击合并及居中按钮合并单元格。



- 选择单元格格式对话框合并单元格对齐选项卡



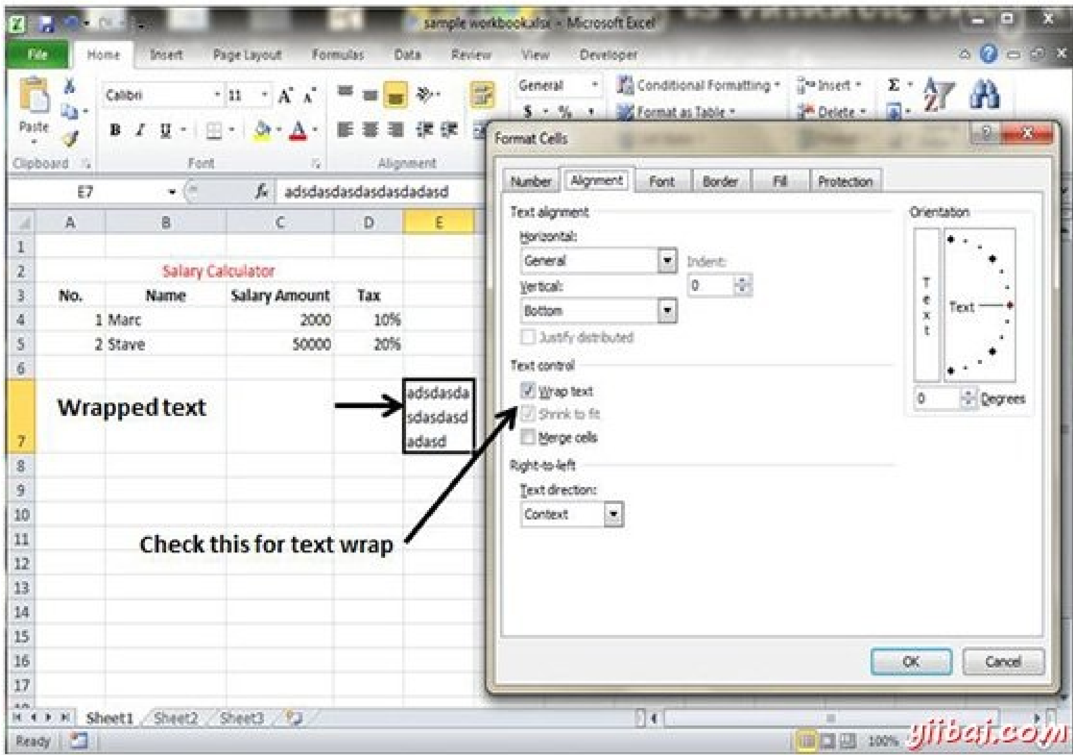
其他选项

主页»对齐方式组»合并与控制中心包含一个下拉列表，这些额外的选项：

- 合并多个：当选择多行范围，该命令将创建多个合并单元格 - 每个行。
- 合并单元格：合并选定单元，而不应用中心属性。
- 取消合并单元格：取消合并选定的单元格。

换行和缩小以适合

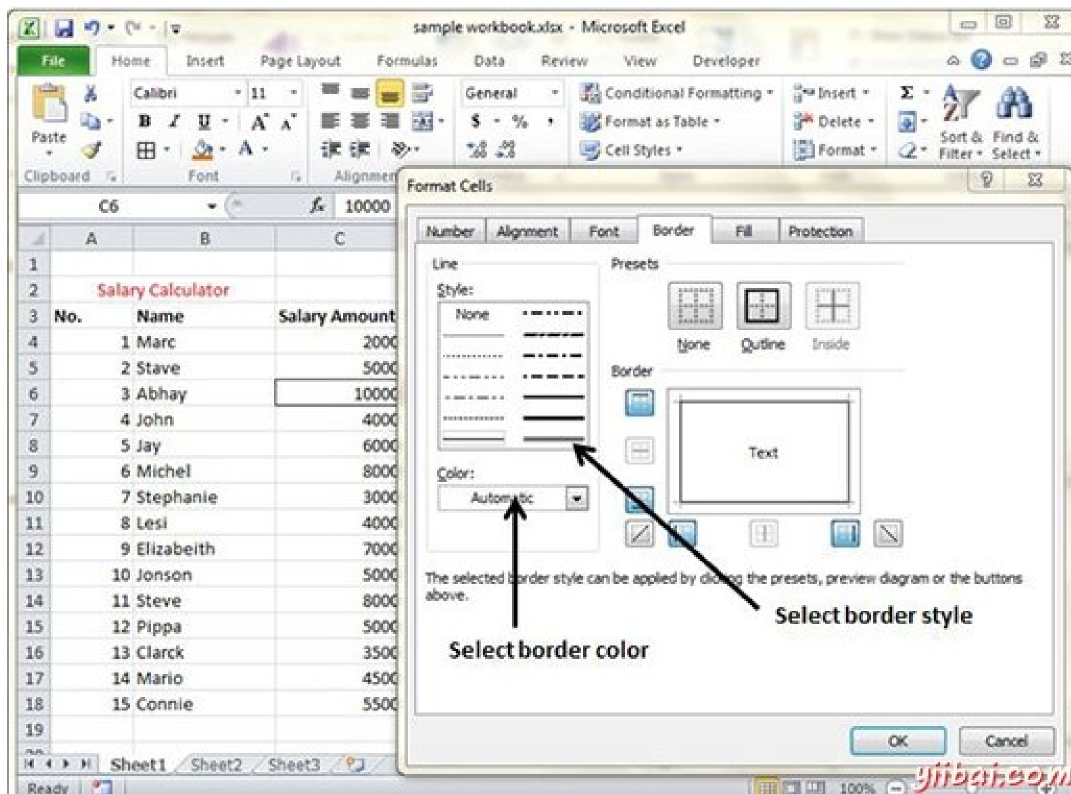
如果您有文字太宽适合列宽，但不希望该文本延到相邻的单元格， 您可以使用环绕文本选项或收缩以适应选项，以适应该文本。


sample workbook.xlsx - Microsoft Excel
Salary Calculator
No.
Name
Salary Amount
Tax
1 Marc
2000
10%
2 Stave
50000
20%
Wrapped text
adsdasda
sdasdasd
adasd
Check this for text wrap
Format Cells
Number
Alignment
Font
Border
Fill
Protection
Text alignment
Horizontal:
General
Indent:
Vertical:
Bottom
Justify distributed
Text control
Wrap text
Shrink to fit
Merge cells
Right-to-left
Text direction:
Context
Orientation
Text
Degrees
OK
Cancel
Sheet1
Sheet2
Sheet3
Ready
yiibai.com

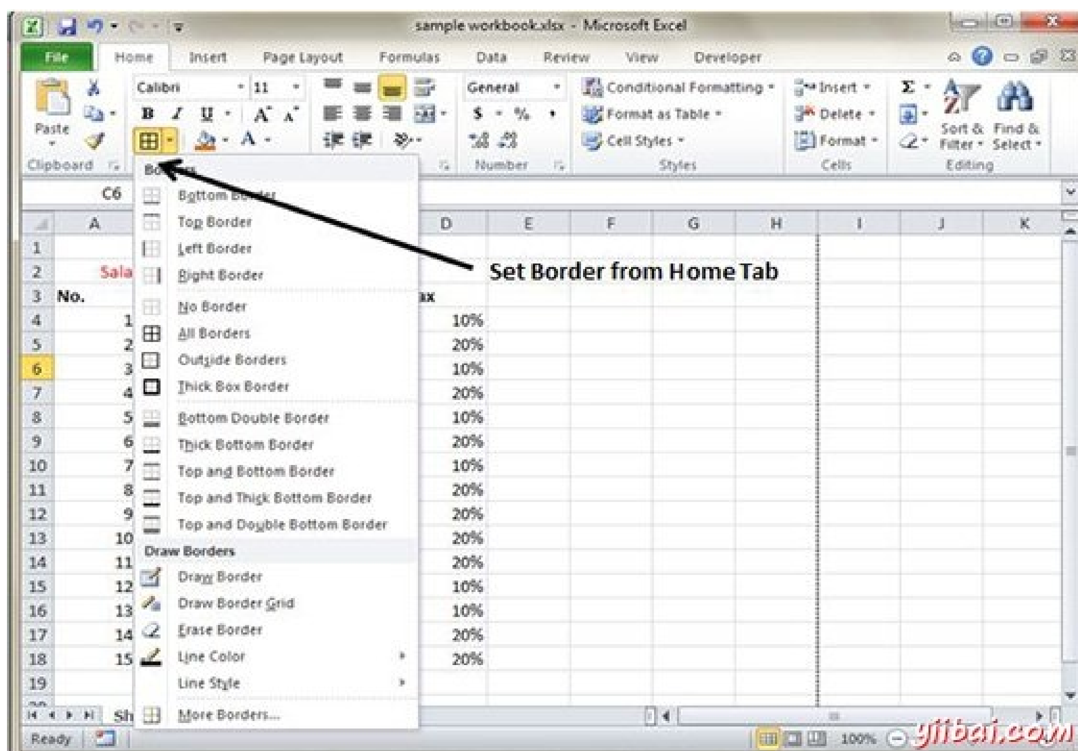
Excel边框和色调 - Excel教程

应用边框

MS Excel中，您可以将边框应用到单元。对于应用边框选择的单元格区域单击鼠标右键»设置单元格格式»边框选项卡»选择边框样式

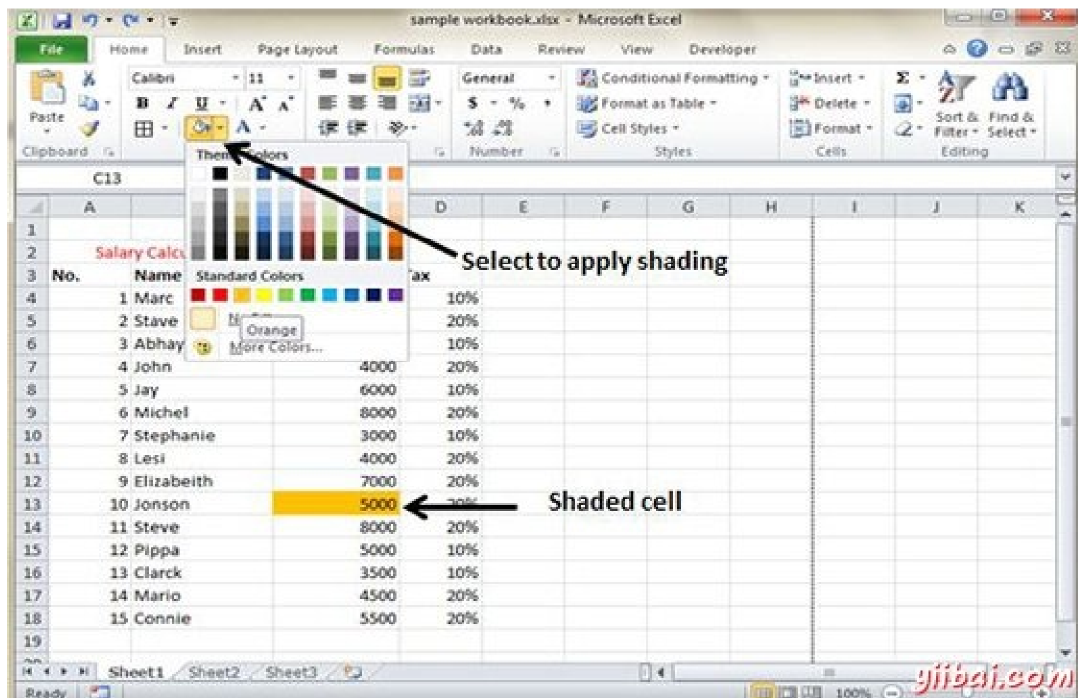


然后，您可以通过主页选项卡应用边框»字体组»边框应用



使用阴影

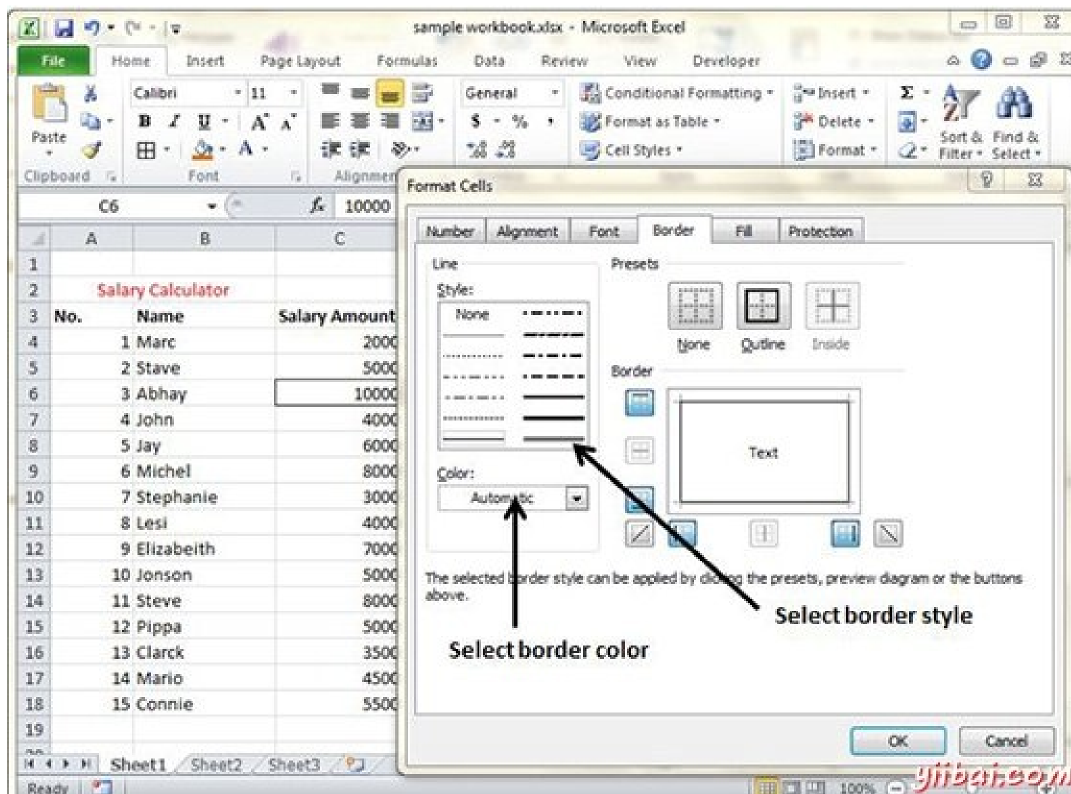
您可以添加阴影从单元格的主页选项卡»字体组»选择颜色



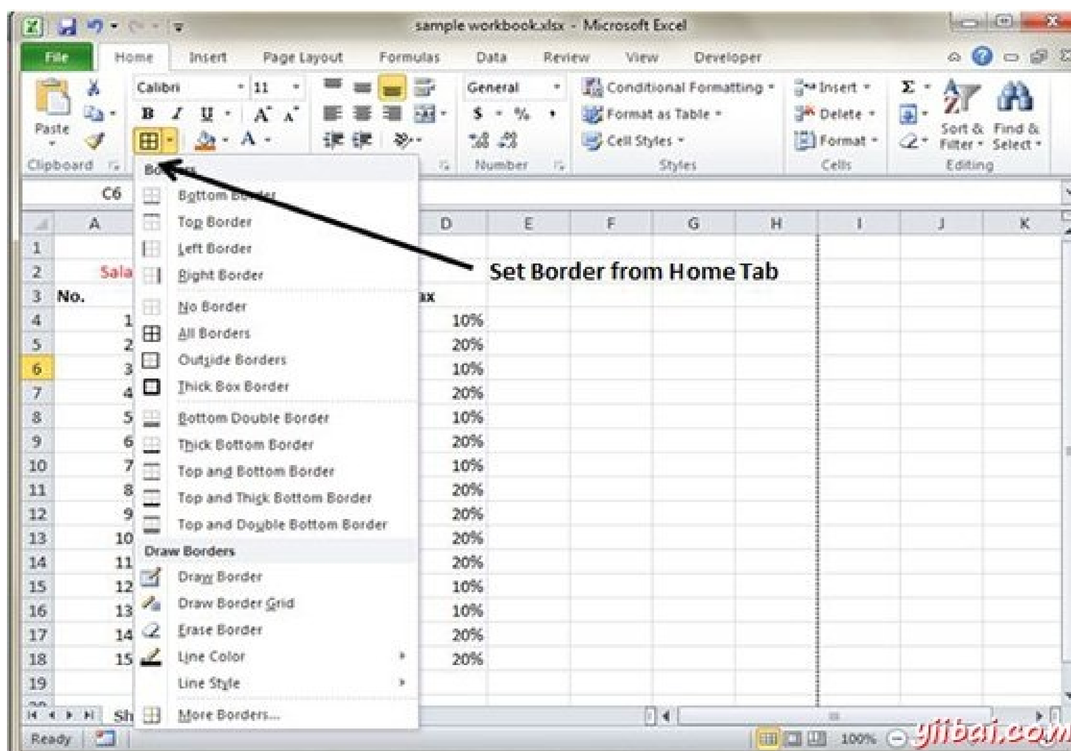
Excel边框和色调 - Excel教程

应用边框

MS Excel中，您可以将边框应用到单元。对于应用边框选择的单元格区域单击鼠标右键»设置单元格格式»边框选项卡»选择边框样式

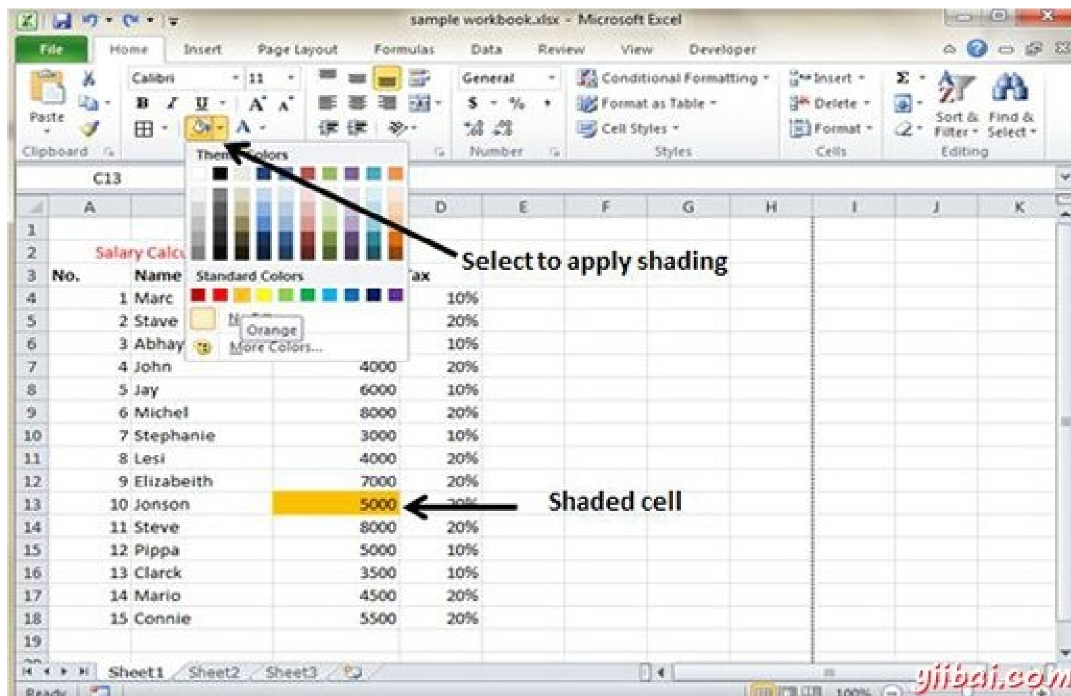


然后，您可以通过主页选项卡应用边框»字体组»边框应用



使用阴影

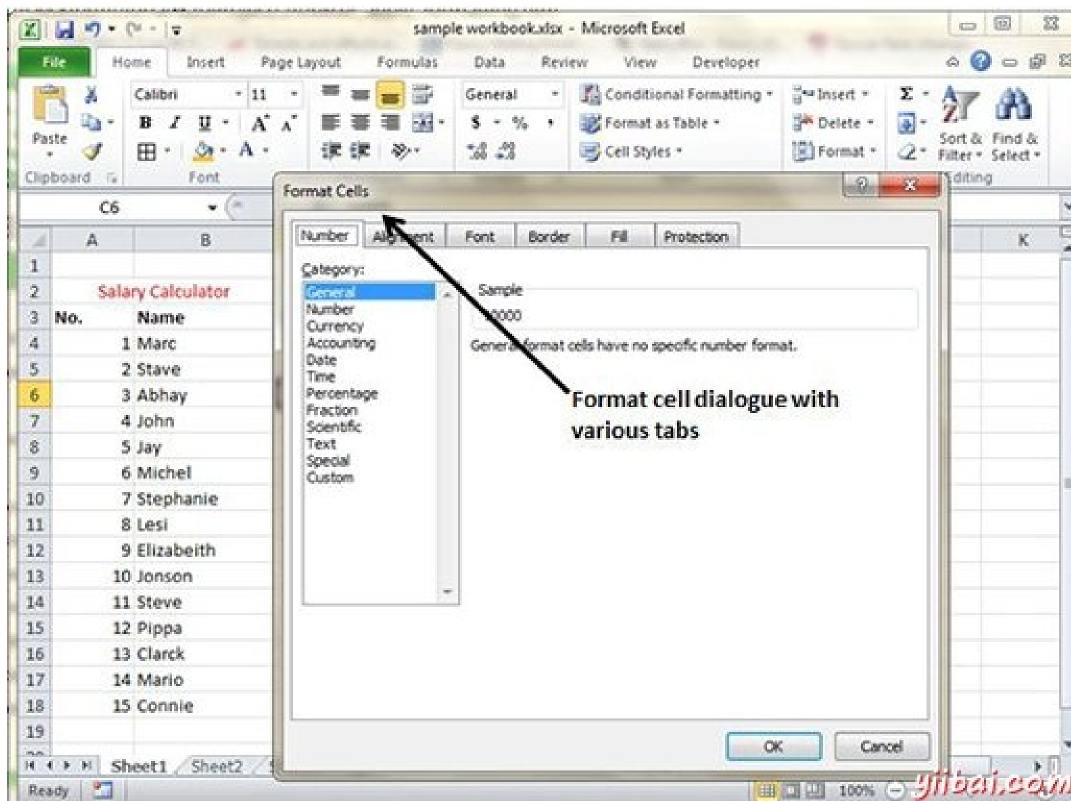
您可以添加阴影从单元格的主页选项卡»字体组»选择颜色



Excel应用格式化 - Excel教程

格式化单元格

在MS Excel中，你可以应用格式单元格的单元格或范围内右键点击»设置单元格格式»选择标签。有各种标签可用如下



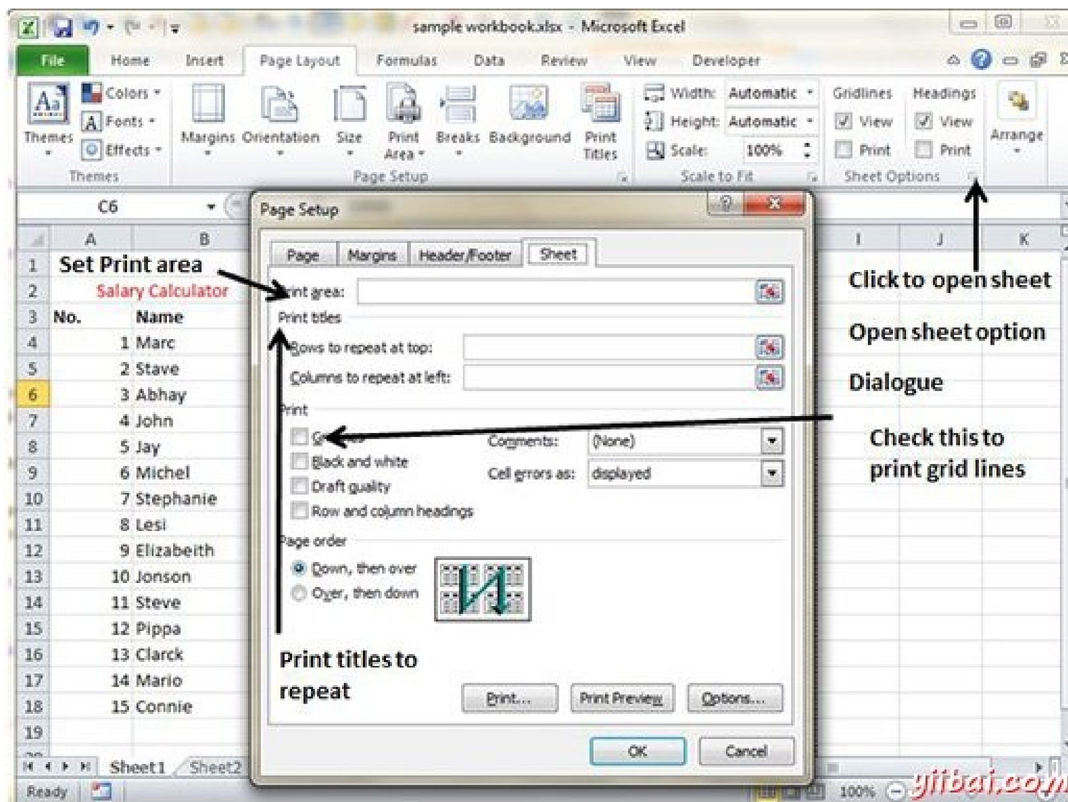
替代放置背景

- 数值：可以设置单元格的格式取决于单元格内容。查找教程在MS Excel - 设置单元类型
- 对齐方式：可以在此选项卡中设置文本的对齐方式。查看教程在MS Excel文本对齐方式
- 字体：可以设置此选项卡上的文字字体。查看教程在MS Excel设置字体
- 边框：可以设置单元格边框与此标签。查看教程在MS Excel边框和色调
- 填充：可以使用此选项卡中设置单元格填充。查看教程在MS Excel边框和色调
- 保护：可以设置该选项卡中单元格保护选项。

Excel表选项 - Excel教程

Excel表选项

MS Excel提供了各种工作表用于打印用途等通常单元格的选择。如果你想打印输出，包括网格线，选择页面布局»表选项组»网格线»检查打印



在表选项对话框选项

- 打印区域：可以设置打印区域使用此选项。
- 打印标题：您可以设置标题出现在为行的顶部和左侧的列。
- 打印：
    - 网格线：以网格线打印时出现在工作表。
    - 黑与白：选中此复选框将彩色打印机打印图表黑色和白色。
    - 草稿：选中此复选框以打印使用打印机的草稿质量设置图表。
    - 行和列标题：选中此复选框有行和列标题打印。
- 页面顺序：

- 向下，然后经过：它打印下来的网页，然后再右页。
- 经过，再向下：它打印右页，然后再来找打印下来的网页。

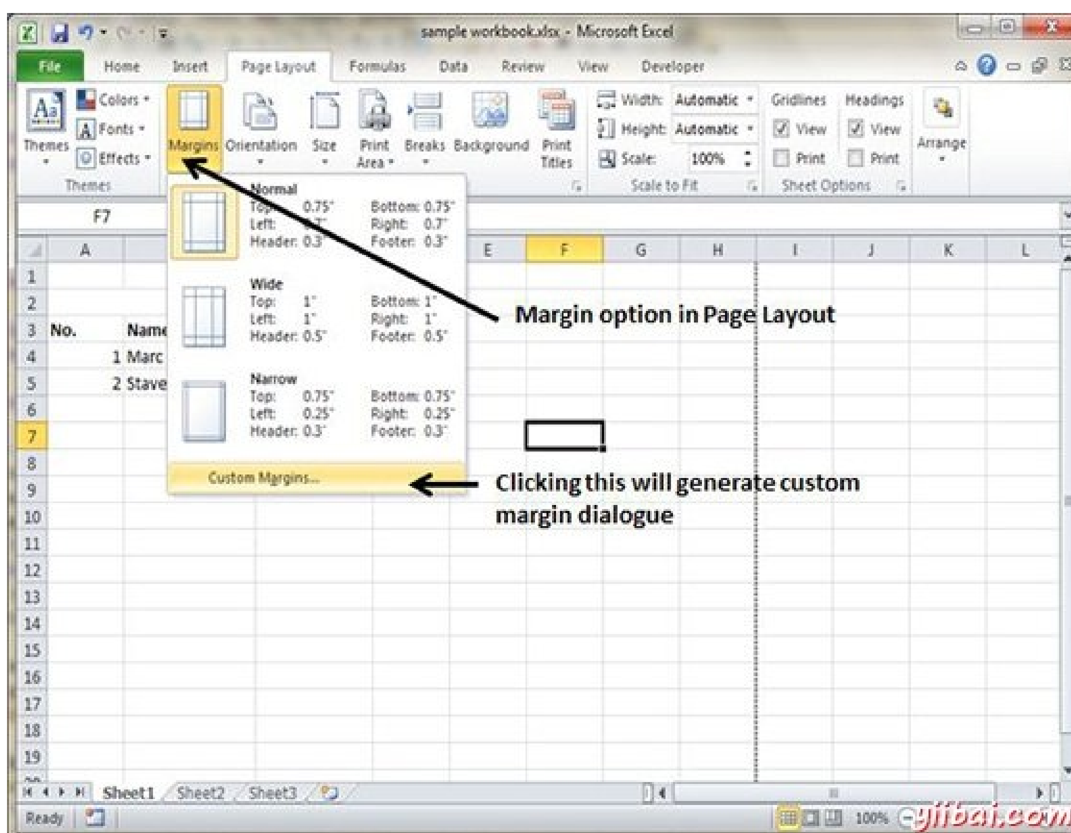
Excel调整边距 - Excel教程

边距

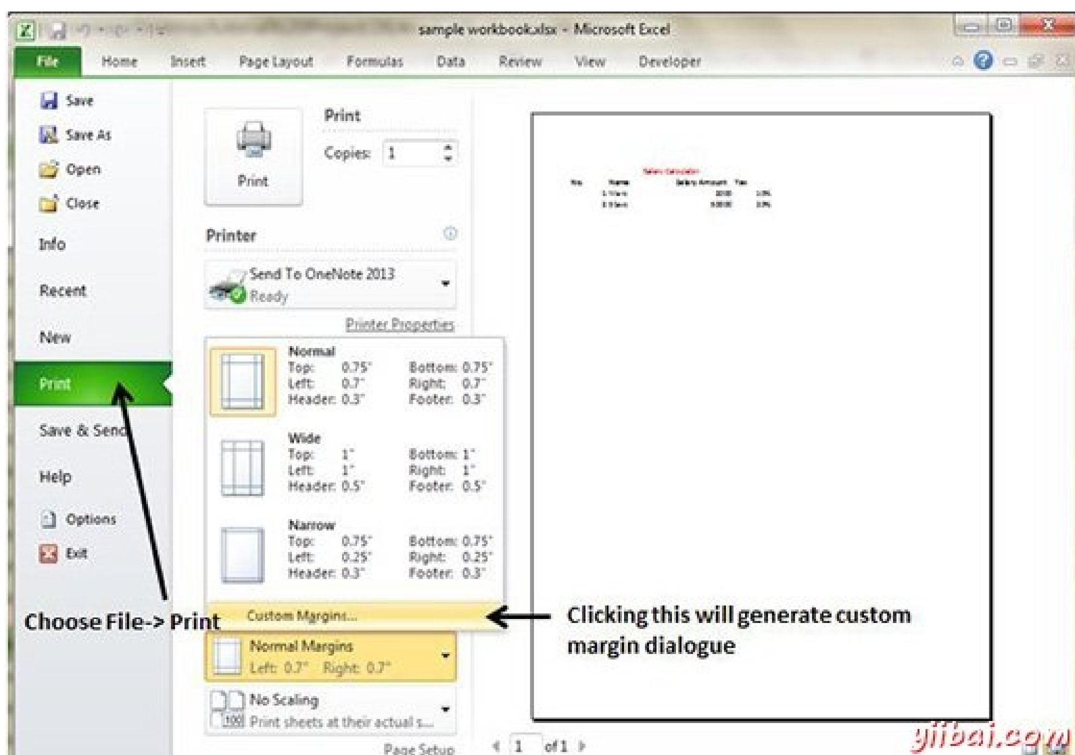
边距是沿着侧面，顶部，和一个打印页的底部的未打印区域。在MS Excel的所有打印页面上有相同的边距。您不能指定不同的边距在不同的页面。

您可以通过多种方式设置页边距，如下

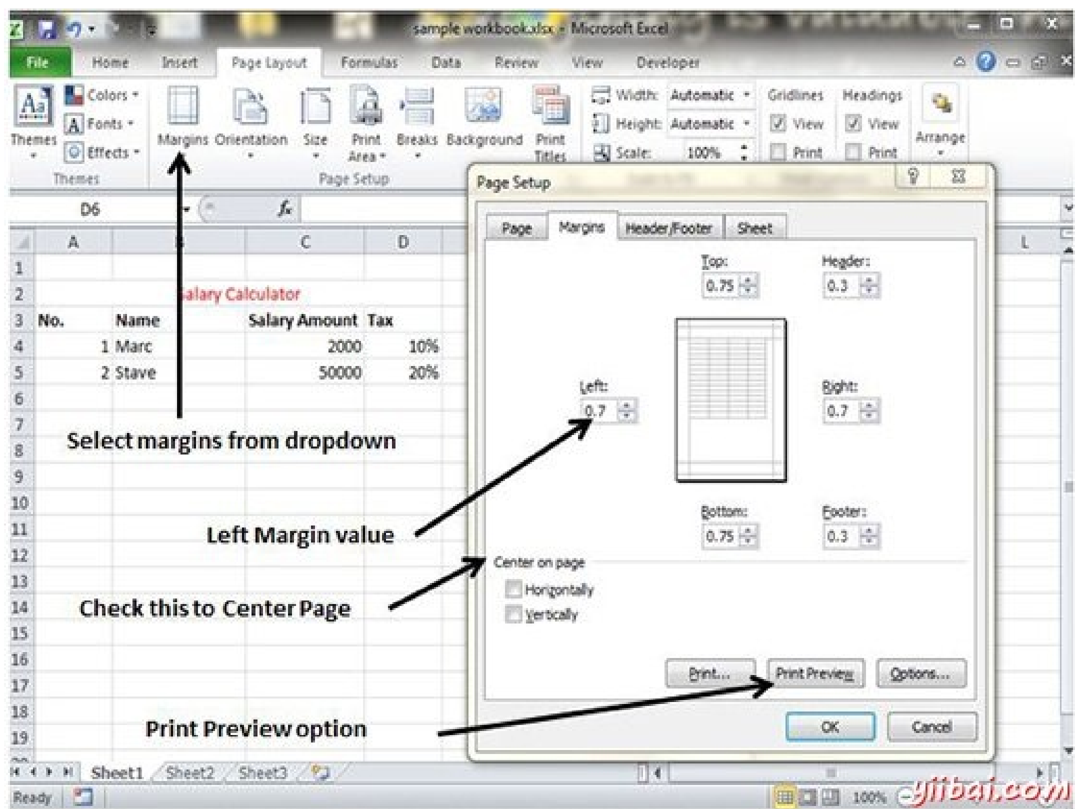
- 选择页面布局»页面设置»边距下拉列表中，您可以选择普通，宽，窄，或自定义设置。



- 这些选项也可当你选择文件»打印。



如果没有这些设置做这项工作，选择自定义边距以显示页面设置对话框中的边距选项卡，如下图所示。



在页面中心

默认情况下，Excel中对齐顶部和左侧边距打印的页面。 如果你想输出进行垂直或水平居中，选择在中心相应的复选框上边距选项卡页部分，如图上面的截图。

Excel页面方向 - Excel教程

页面方向

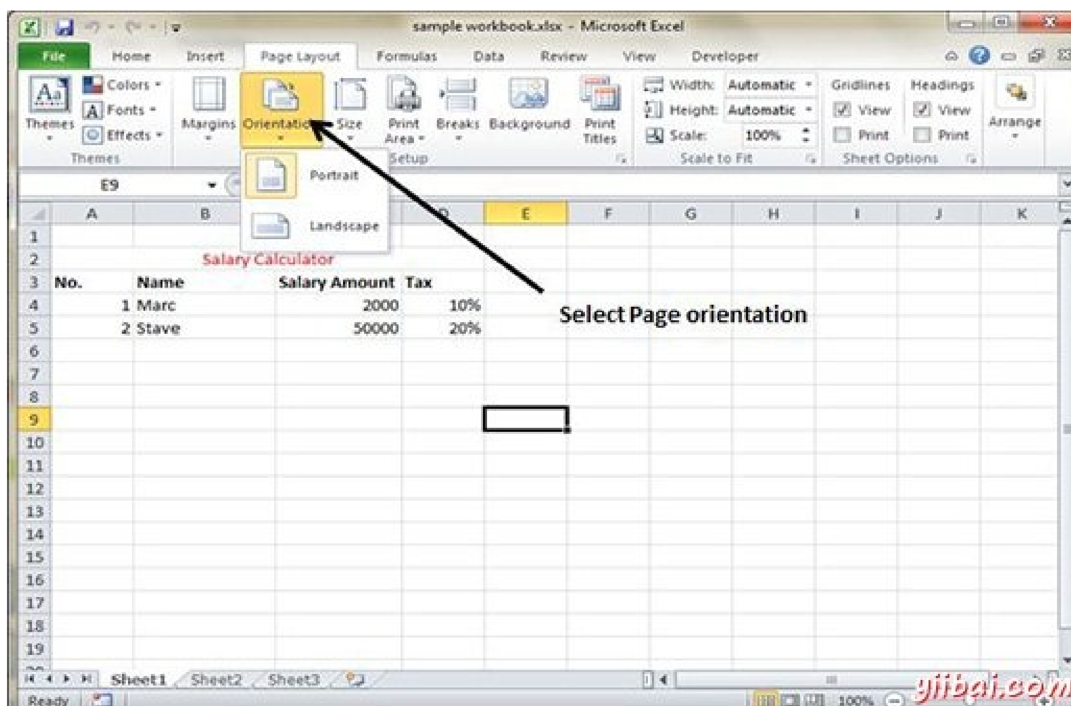
页面方向是指如何输出被打印在页面上。如果你改变了方向，屏幕上的分页符自动调整以适应新的纸张方向。

页面方向类型

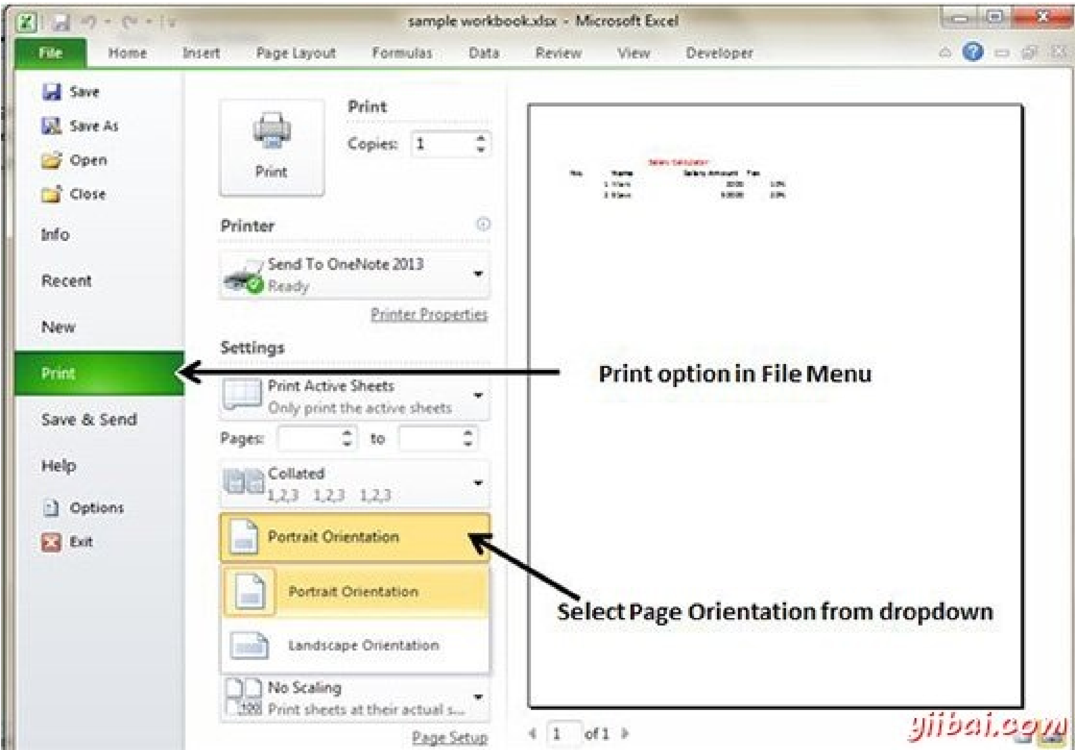
- 纵向：纵向打印页面高度(默认值)。
- 横向：横向打印宽网页。当你有一个广泛的不适合垂直方向的页面上横向是非常有用的。

更改页面方向

- 选择页面布局»页面设置»»方向纵向或横向



- 选择文件»打印


sample workbook.xlsx - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Save
Save As
Open
Close
Info
Recent
New
Print
Save & Send
Help
Options
Exit
Print
Copies: 1
Printer
Send To OneNote 2013
Ready
Printer Properties
Settings
Print Active Sheets
Only print the active sheets
Pages:
to
Collated
1,2,3 1,2,3 1,2,3
Portrait Orientation
Portrait Orientation
Landscape Orientation
No Scaling
Print sheets at their actual s...
Page Setup
Print option in File Menu
Select Page Orientation from dropdown
1 of 1
yiibai.com

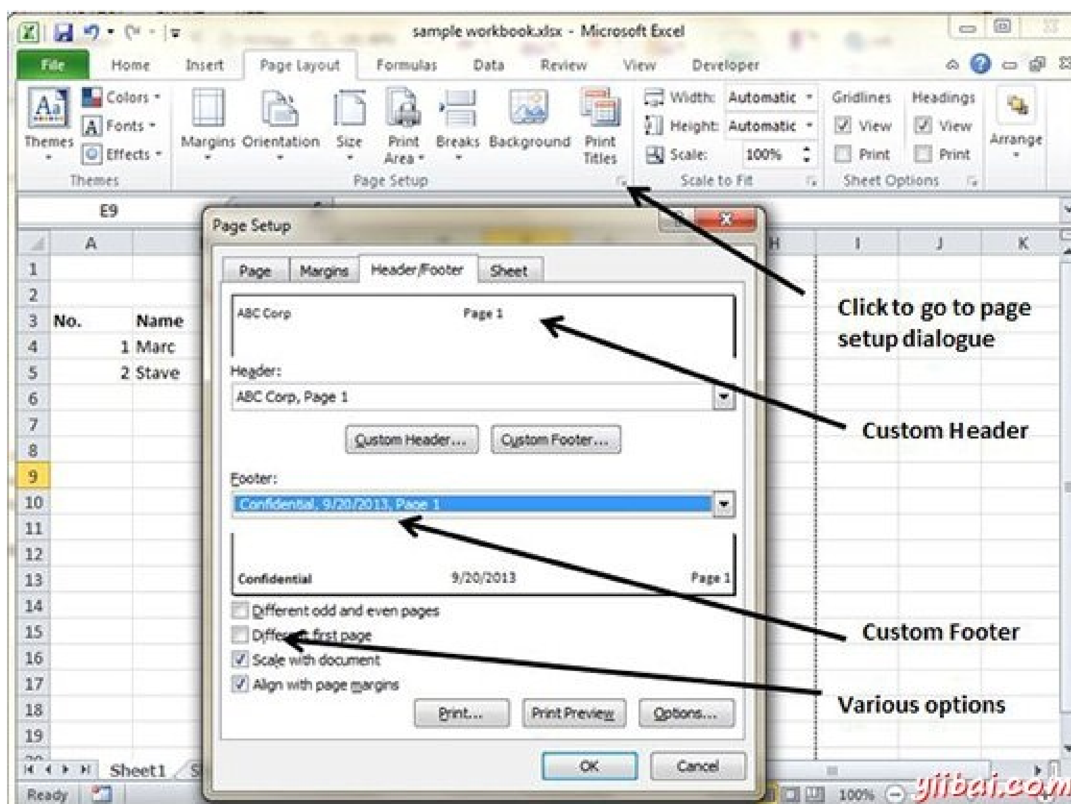
Excel页眉和页脚 - Excel教程

页眉和页脚

页眉是出现在每个打印页的顶部信息和页脚是出现在每个打印页面的底部信息。 默认情况下，新的工作簿没有页眉或页脚。

添加页眉和页脚

- 选择页面设置对话框»页眉或页脚选项卡



你可以选择预定义的页眉和页脚，或创建自定义

- **&[Page]** : 显示页面码
- **&[Pages]** : 显示要打印的总页数
- **&[Date]** : 显示当前的日期
- **&[Time]** : 显示当前时间
- **&[Path]&[File]** : 显示工作簿的完整路径和文件名
- **&[File]** : 显示工作簿的名称

- **&[Tab]：**显示表的名字

其它页眉和页脚选项

当一个页眉或页脚在页面布局视图中选择，页眉和页脚»设计»选项组包含让您指定的其他选项控制：

- 不同的第一页：选中该指定不同的页眉或页脚为第一打印页上。
- 不同奇数和偶数页：选中该指定不同的页眉或页脚奇数页和偶数页。
- 缩放文件：如果选中，在页眉和页脚的字体大小会大。因此，如果打印文档时缩放。启用此选项，默认情况下。
- 与页边距对齐：如果选中，左侧页眉和页脚将与左边缘对齐，右页眉和页脚将与右页边距对齐。启用此选项，默认情况下。

Excel插入分页符 - Excel教程

分页符

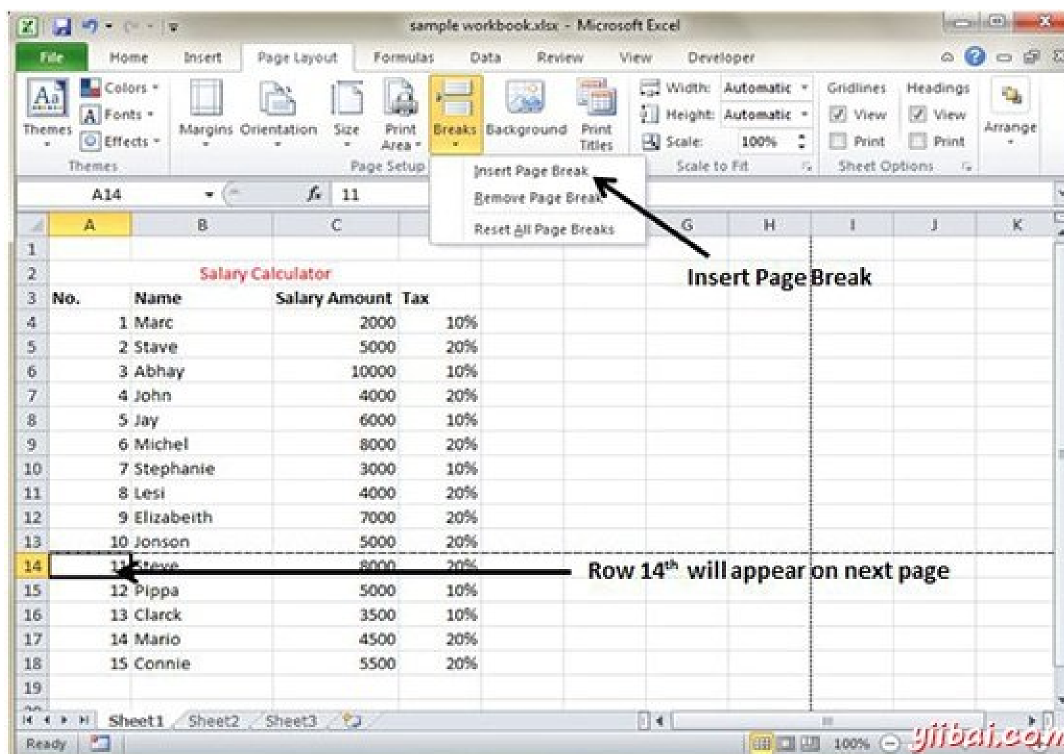
如果你不想行打印页本身或者你不想表格标题行是一个页面上的最后一行。 MS Excel为您提供了精确的控制分页符。

MS Excel会自动处理分页，但有时可能需要强制分页符垂直或水平，所以，此报告打印你想要的方式。

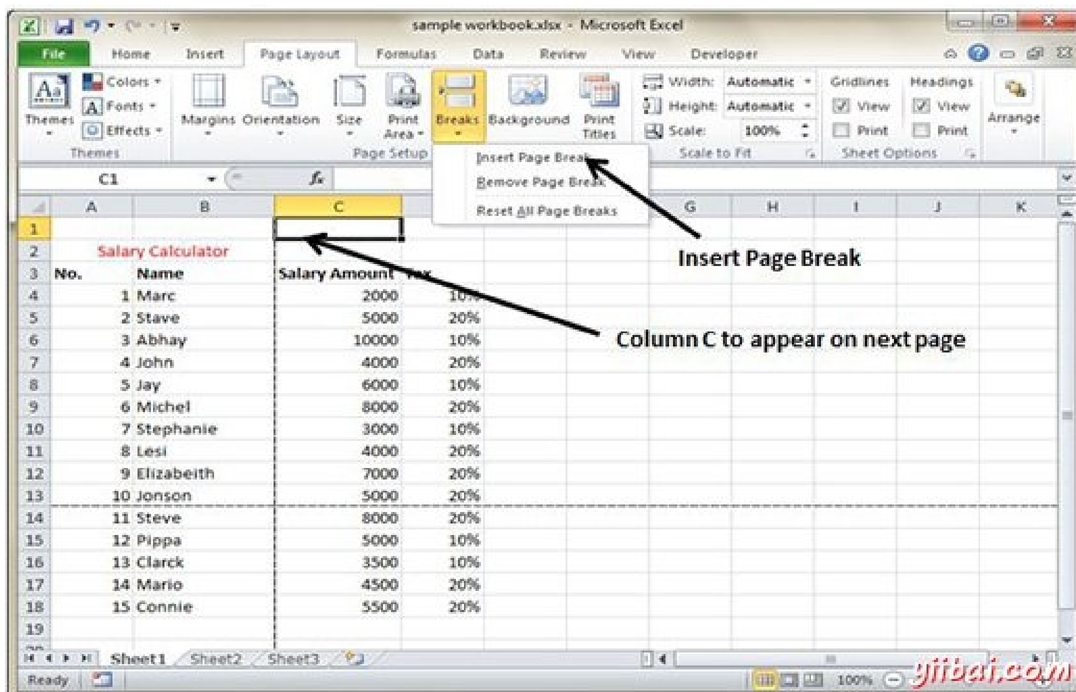
例如，如果工作由几个不同的部分，可能想在另一张纸上打印每个部分。

插入分页符

插入水平分页符：例如，如果想要行14是一个新的页面，第一行选择单元格A14。然后选择页面布局»页面设置组»分页符»插入分页符。



插入垂直分页符在这种情况下，要确保将指针放在第1行。选择页面布局»页面设置»符»插入分页符创建分页符。



删除分页符

- 删除添加了的一个分页符：将单元格指针到第一行下方的手动分页符，然后选择页面布局»页面设置»符»删除分页符。
- 删除所有手动分页符：选择页面布局»页面设置»分页符»重置所有分页符。

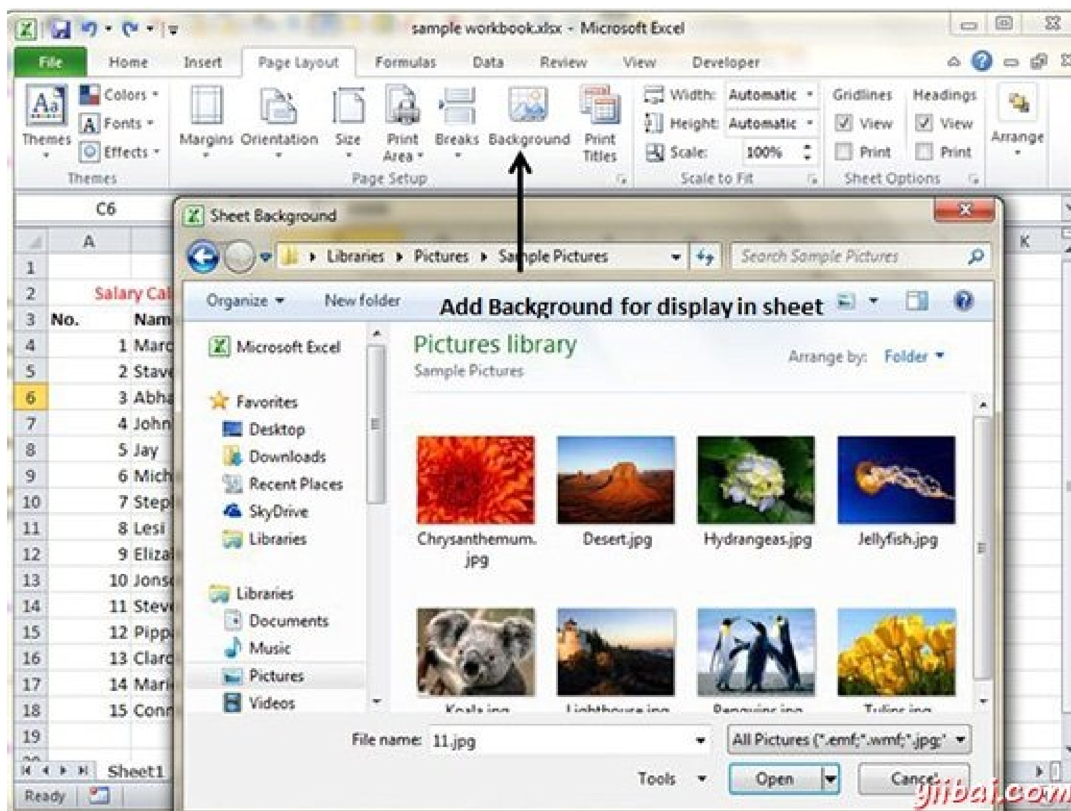
Excel设置背景 - Excel教程

背景图片

如果想你的打印输出的背景图像，然后不幸的是，不能。 可能已经注意到了页面布局»页面设置»背景命令。此按钮将显示一个对话框，让你选择一个图像显示为背景。放置这种控制在其他打印相关的命令是很大的误导。放置在一个工作表背景图片永远不会打印。

替代放置背景

- 您可以将图形，艺术字，或者你的工作表上的照片，然后调整其透明度。然后将图像复制到所有打印页面。
- 可以插入页眉或页脚的对象。



Excel冻结窗格 - Excel教程

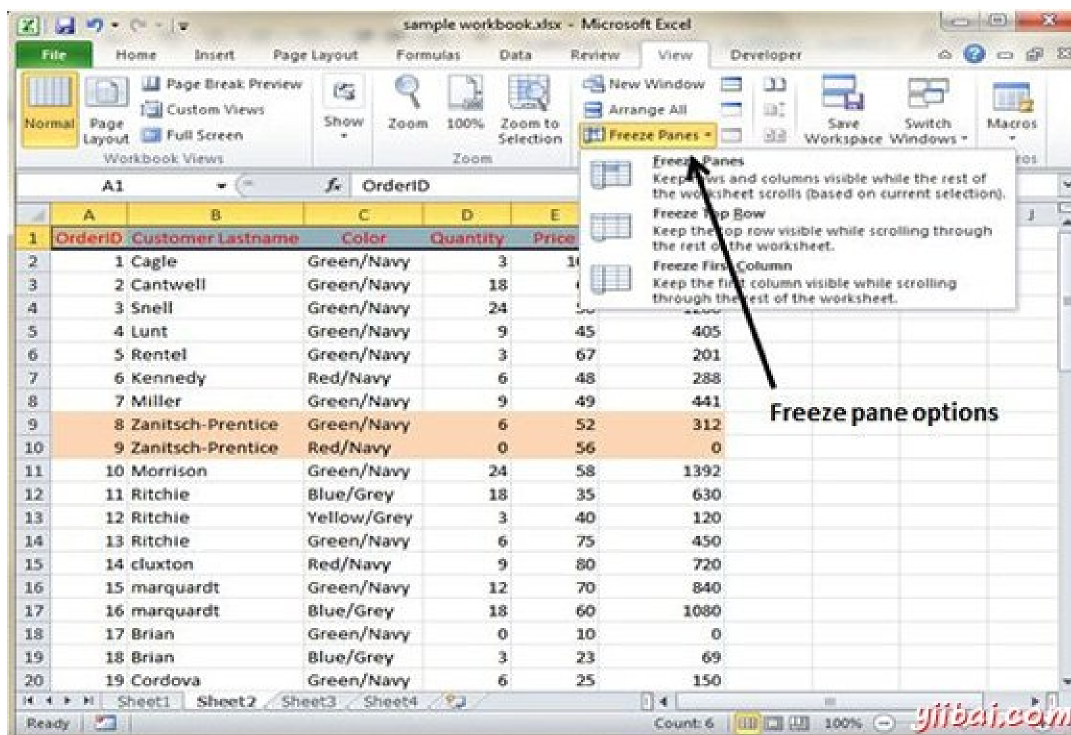
冻结窗格

如果你设置了行或列标题在工作表中，当向下滚动或向右这些标题将不可见。MS Excel提供了一个方便的解决这个问题，冻结窗格。冻结窗格保持标题可见，在滚动工作表时。

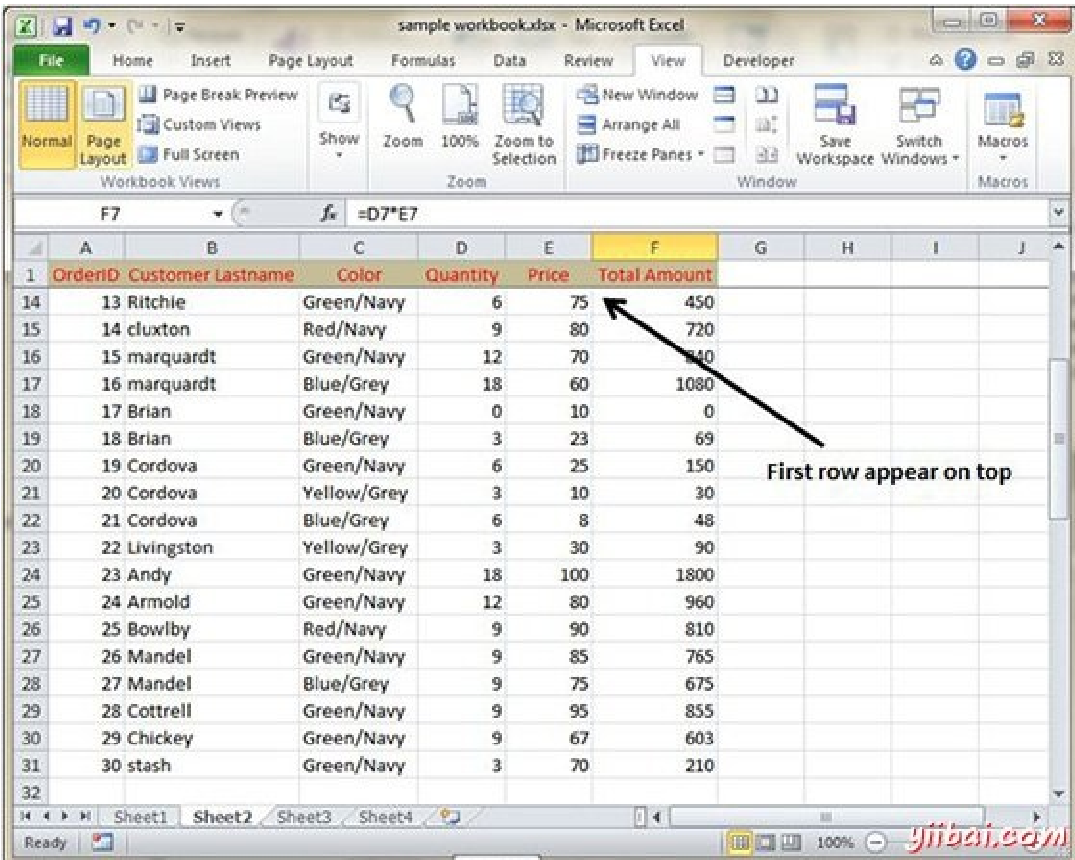
使用冻结窗格

请按照以下步骤做冻结窗格

- 选择第一行或第一列或行下面是要冻结或列右到要冻结区域
- 选择查看标签»冻结窗格
- 选择合适的选项
  - 冻结窗格：要冻结单元区域
  - 冻顶行：冻结工作表的第一行
  - 冻结第一列：冻结工作表的第一列



- 如果您选择冻结顶行，可以看到第一行还滚动后出现在顶部。请参阅下面的屏幕截图



取消冻结窗格

解冻窗格中选择查看标签»取消冻结窗格

Excel条件格式 - Excel教程

条件格式

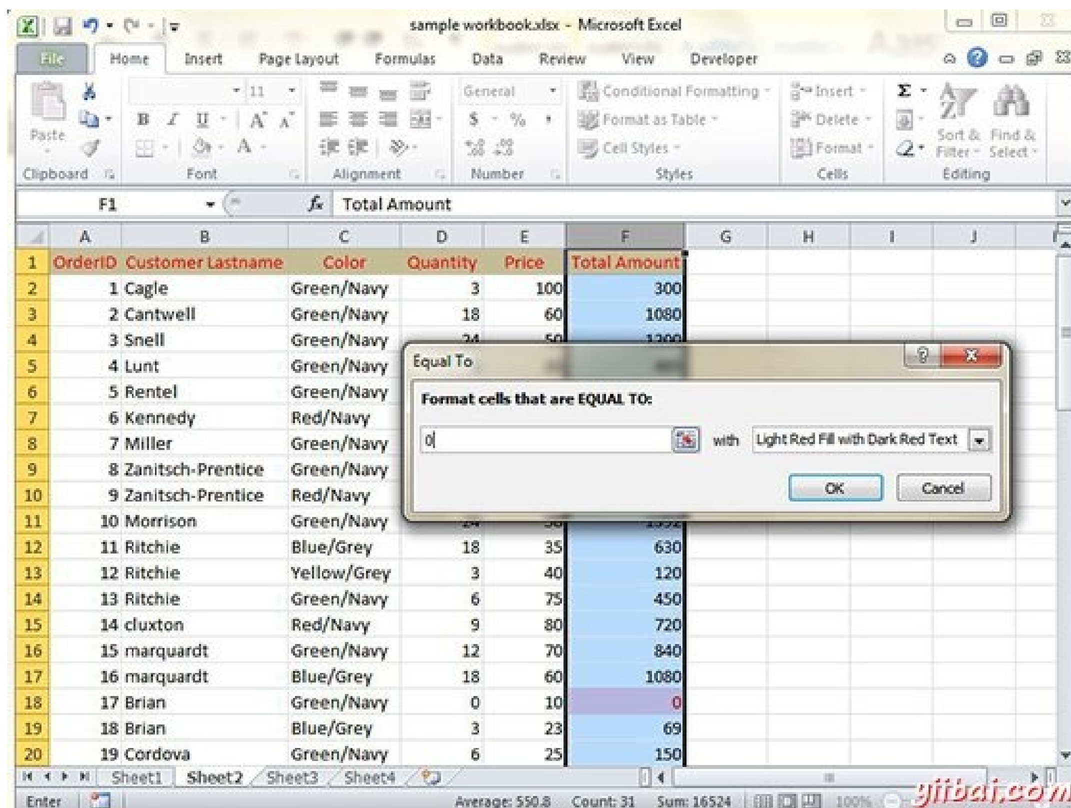
MS Excel 2010的条件格式功能，可以格式化值的范围，使外部一定的限度值，会自动格式化。

选择主页选项卡»样式组»条件格式下拉列表。

各种条件格式选项

- 高亮单元格规则：它带有多种选项的延续菜单用于定义高亮显示单元在包含某些值，文本或日期，或具有值的比的特定值更大或更小，或选择格式化规则值的一定范围内。

假设你想找到具有单元格数量0，并标记它们为红色。选择的单元格»主页选项卡»条件格式下拉范围»高亮度小区规则»等于



单击确定单元格具有零值后，被标记为红色。

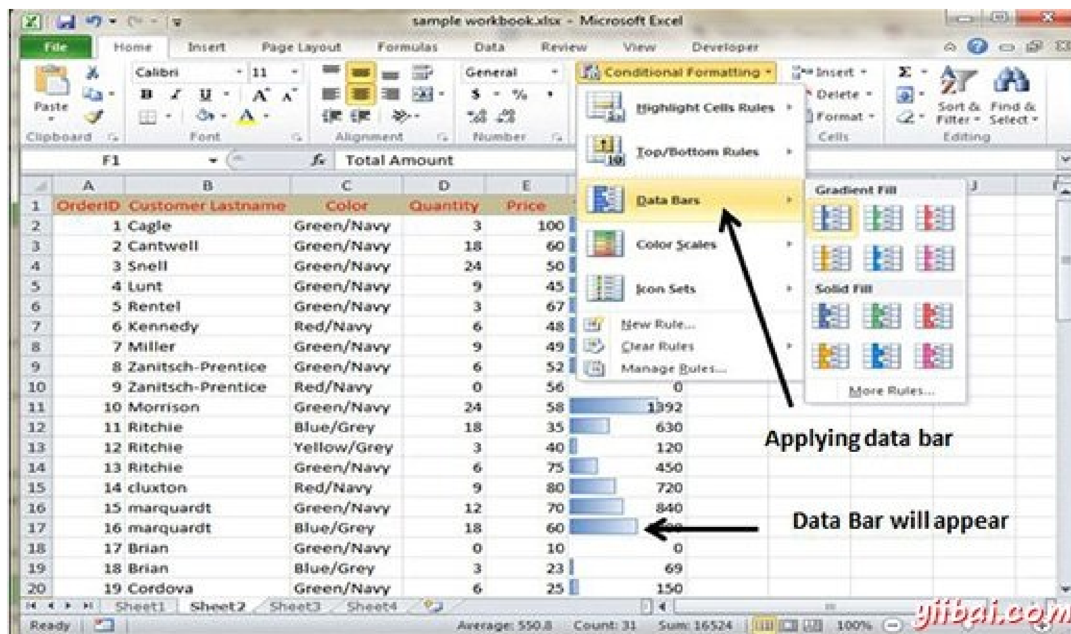
- 顶部/底部规则：它带有多种选项来定义，突出的顶部和底部值，百分比及上面和下面的平均值在小区选择格式化规则的延续菜单。

假设想强调前10%的行，你可以做到这一点与这些顶部/底部规则

- 数据栏：它会打开可以应用到小区选择相对于彼此通过单击数据栏缩略图来表示它们的

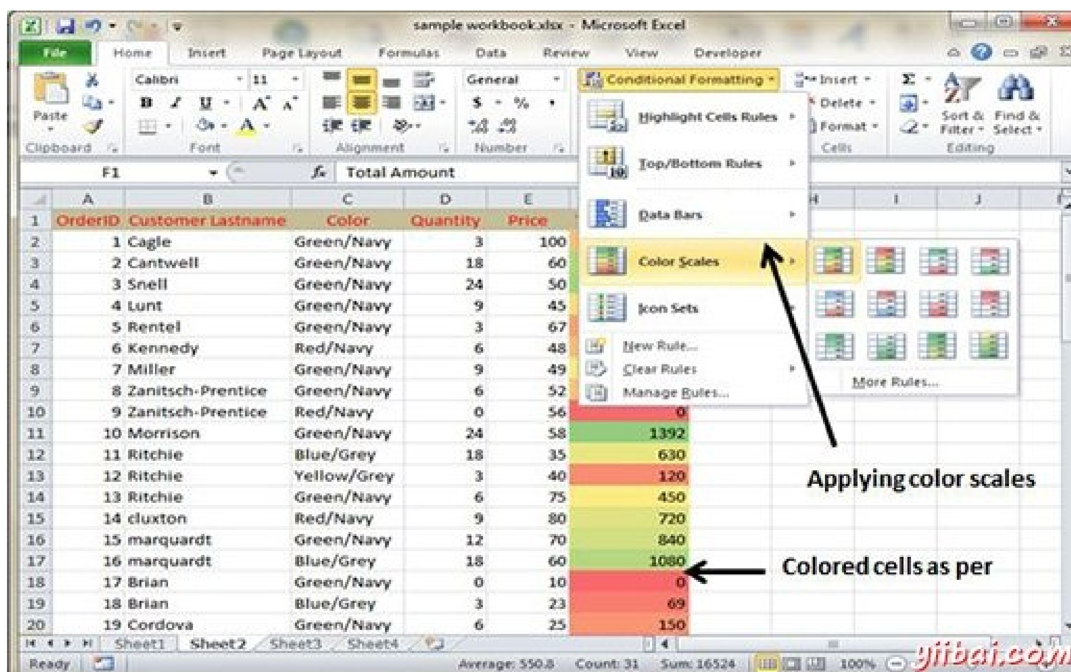
值在不同颜色的数据栏的调色板。

有了这个条件格式数据栏会出现在每个单元中。



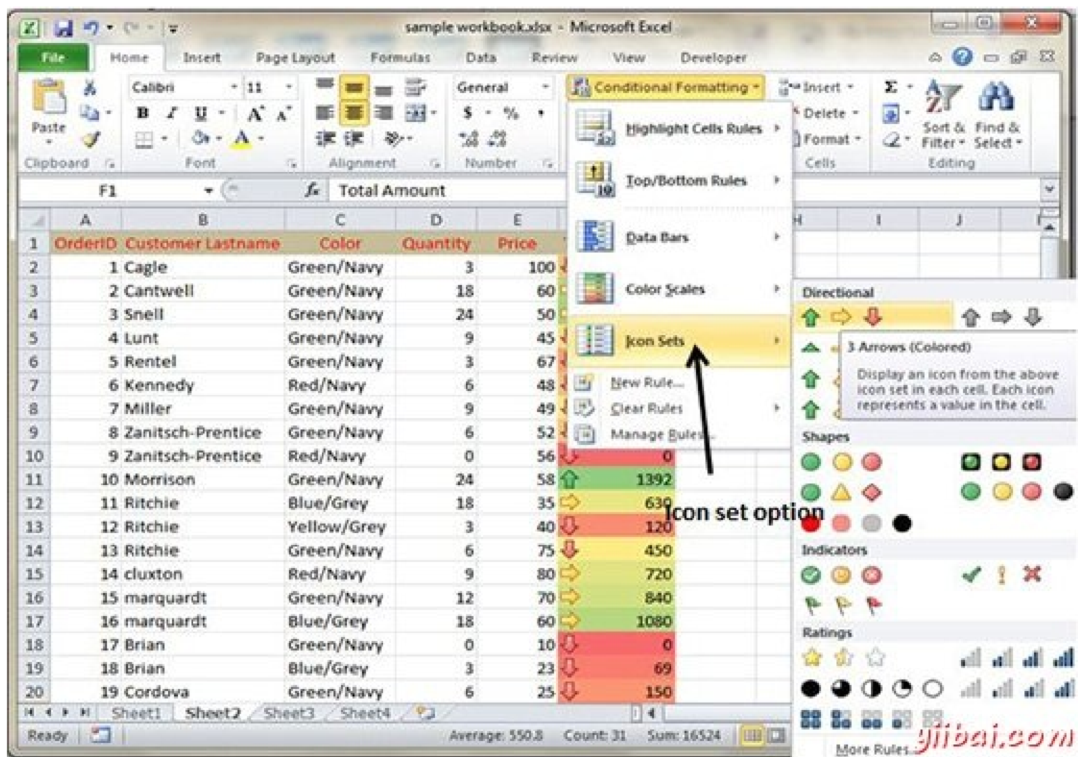
- 色标：它打开不同的三和二色鳞片的调色板，可以应用到单元格选择相对于彼此通过点击色标缩略图来表示它们的值。

请参见下面的截图具有色标应用条件格式。



- 图标设置：它会打开不同组的图标，你可以应用到单元格选择相对于彼此通过点击图标设置为显示其值的调色板。

请参见下面的截图具有图标设置应用于条件格式。



- 新规则：它打开了新的格式规则对话框，在其中定义自定义条件格式规则应用到单元格选择。
- 明确的规则：它打开一个延续菜单，在这里你可以通过点击选中的单元格选项单击整个工作表选项，通过点击该表中删除条件格式规则为单元格选择，对于整个工作表，或仅仅是当前数据表选项。
- 管理规则：它打开条件格式规则管理器对话框，在其中您可以编辑和删除特定的规则，以及通过上移或规则列表框中向下调整自己的规则优先级。

Excel创建公式 - Excel教程

MS Excel公式

公式是工作表的面包和黄油。如果没有公式工作表将数据只是简单的表格表示。公式由被输入到单元格特殊代码。它执行一些计算，并返回一个结果，并显示在单元格中。

公式使用各种运算符和工作表函数具有值和文本工作。公式中使用的值和文本可以位于其他单元格，这使改变数据容易并给出工作表的动态本质。例如，您可以快速改变工作表和公式工作数据。

公式的要素

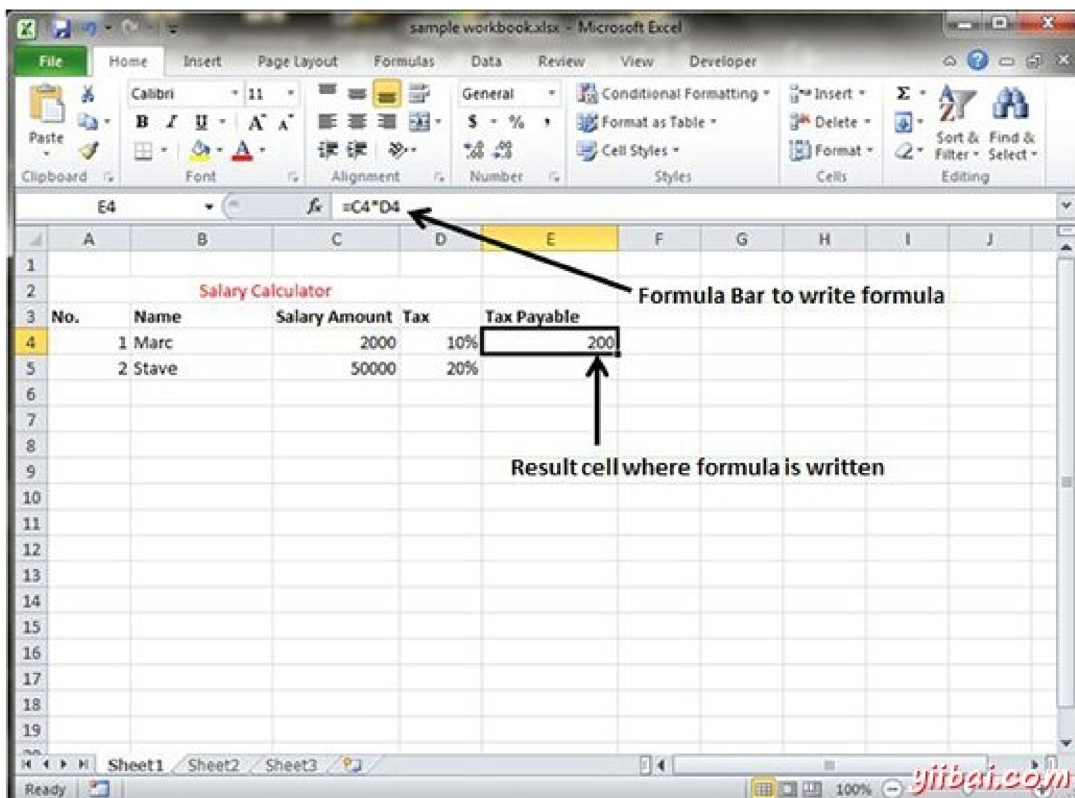
公式可以包含这些元素：

例如：

- 数学运算符，如+(加法)和*(乘法)
    - =A1+A2 相加单元格A1和A2的值。
- 值或文本
    - =200*0.5 200乘以0.5倍。这个公式只使用值，它总是返回相同的结果为100。
- 单元格引用(包括命名的单元格和范围)
    - =A1=C12 比较单元格A1与单元格C12。如果值是相同的，该公式返回TRUE;否则，返回FALSE。
- 工作表函数(如SUMor平均值)
    - =SUM(A1:A12) 在添加范围的值在A1：A12。

创建公式

要创建公式，你需要输入公式栏。公式以“=”号开始。当手动构建公式，您可以键入单元格地址，也可以指向他们在工作表中。使用定点方法提供了单元格地址的公式构建往往更容易，更有效的方法。如果您使用的是内置的功能，通过点击的单元格范围定义函数的参数的函数参数对话框，当您想要使用的单元或拖动。请参阅下面的屏幕截图。



当完成一个公式项，Excel计算的结果，然后将其在工作表内的单元格内显示(公式中的内容，但是，仍然是对公式栏随时单元格是活性可见)。如果你把一个错误，阻止Excel能够计算公式在所有的公式中，Excel会显示一个警告对话框，提示如何来解决这个问题。

Excel复制公式 - Excel教程

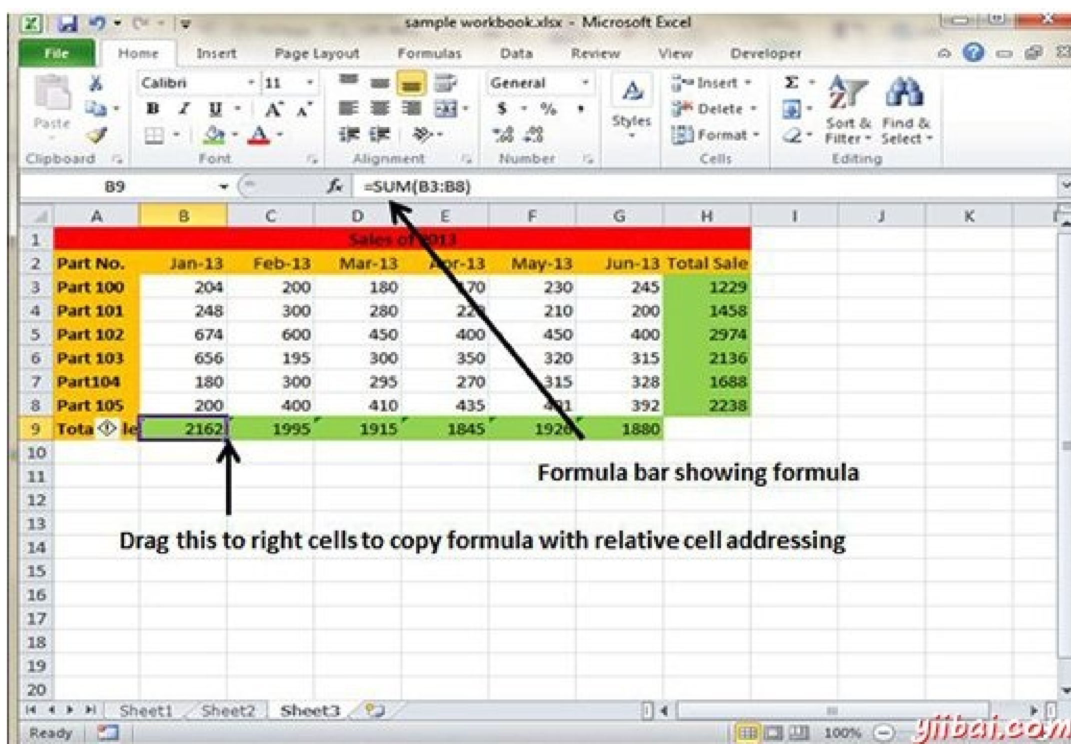
在MS Excel中复制公式

复制公式是你在一个典型的电子表格主要依靠公式执行的最常见的任务之一。当公式使用单元格引用而不是常数值，Excel可以复印原稿公式每一个需要类似的公式地方的任务。

相对单元格地址

MS Excel中它会自动调整，在原来的公式的单元格引用，以适应副本的位置。 它通过称为相对单元格地址的系统，凡在公式更改单元格地址的列引用，以适应新的列位置和行引用改变，以适应他们的新行的位置。

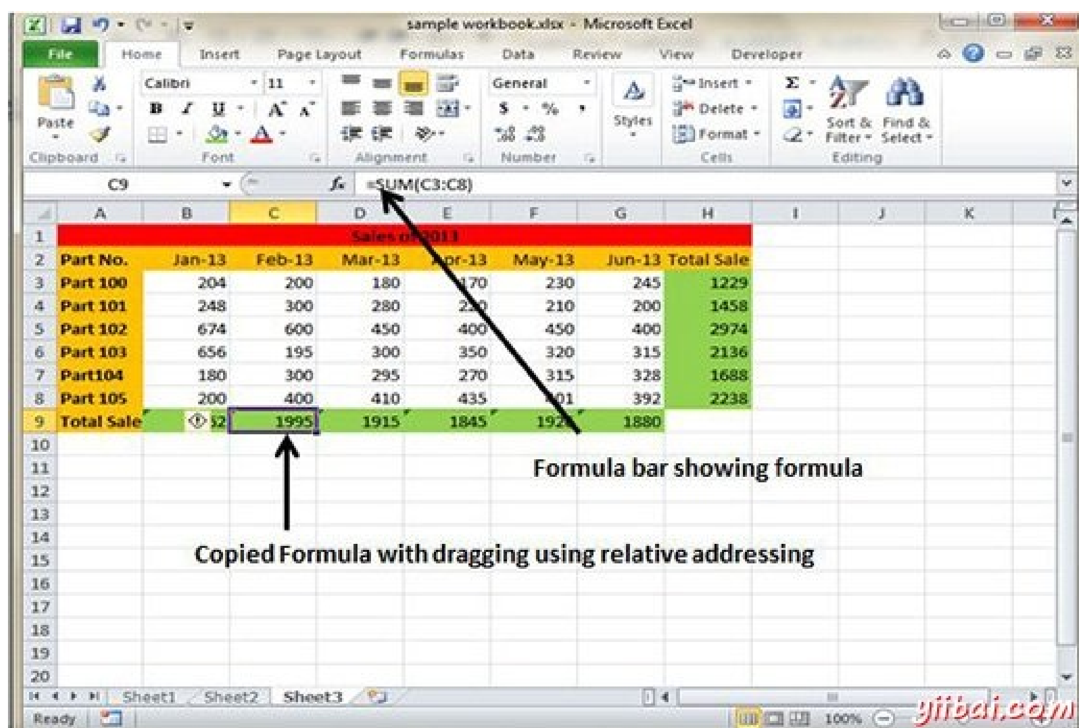
让我们来看看这个例子与帮助。即公式，假设我们希望在最后所有的行总和那么我们会写第一列，我们所需的行的总和为3至8的第9行中。



写公式在第九列后，我们就可以将其拖动到其余列和公式被复制。拖动后我们可以看到公式中的其余列如下。

- column C : =SUM(C3:C8)
- column D : =SUM(D3:D8)
- column E : =SUM(E3:E8)

- column F : =SUM(F3:F8)
- column G : =SUM(G3:G8)



Excel公式参考 - Excel教程

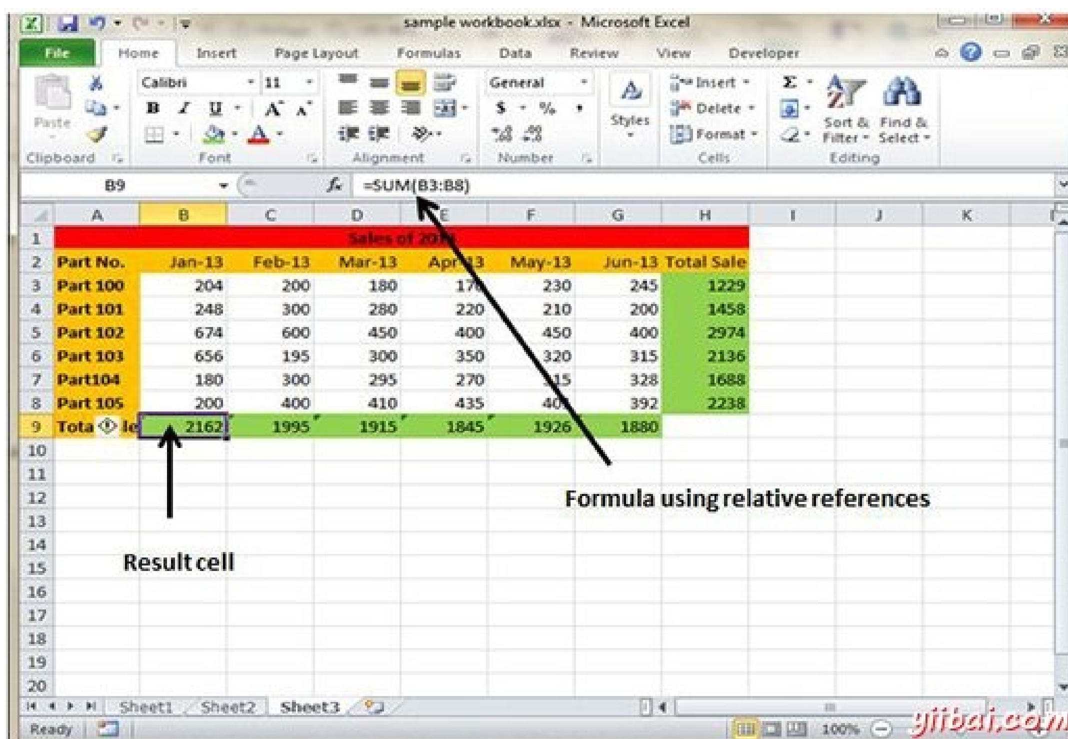
单元格公式引用

创建大多数公式包含对单元格或范围。这些引用使您的公式包含在这些单元格或范围的数据动态地工作。例如，如果你的公式引用单元格C2和更改包含在C2的值，公式结果反映了新的值，并且自动未在公式中使用引用，就需要以改变公式中使用的值来编辑公式本身。

当您使用公式的单元格（或区域）的参考，你可以使用三种类型的引用：相对的，绝对的，和混合引用

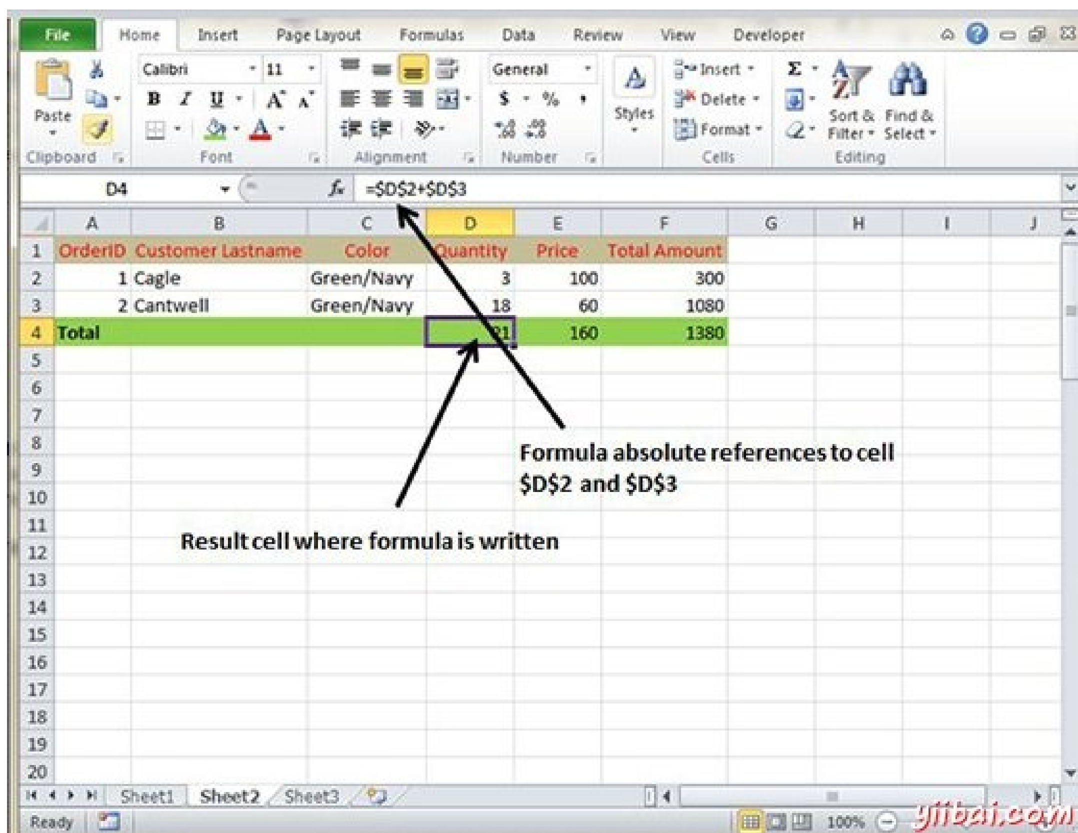
相对单元格引用

当将公式复制到另一个单元格，因为引用实际上从当前行和列偏移的行和列的引用可以改变。默认情况下，Excel创建公式相对单元格引用。



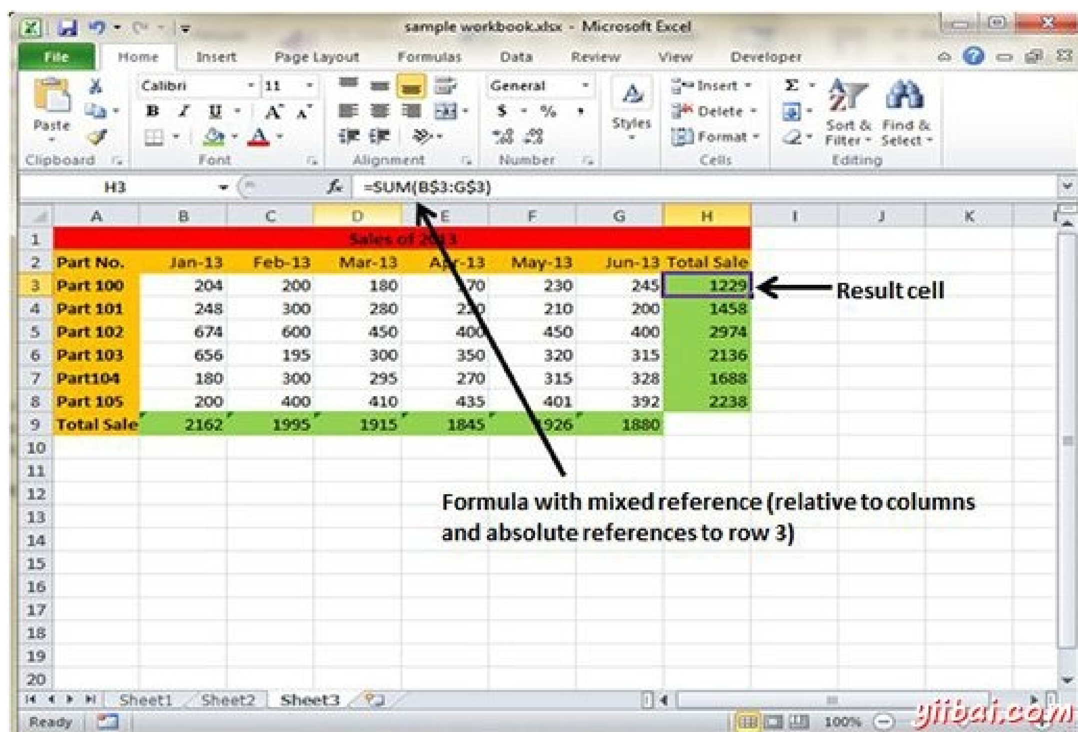
绝对单元格引用

当您复制公式，因为引用的是一个实际的单元格地址的行和列引用不会改变。绝对参考使用其地址的两个美元符号：一个用于列字母，一个用于行数（例如， A5）。



混合单元格引用

无论是行或列的参考都是相对的，而另一个是绝对的。 只有地址部分的一个是绝对的（例如，$A5或A$5）



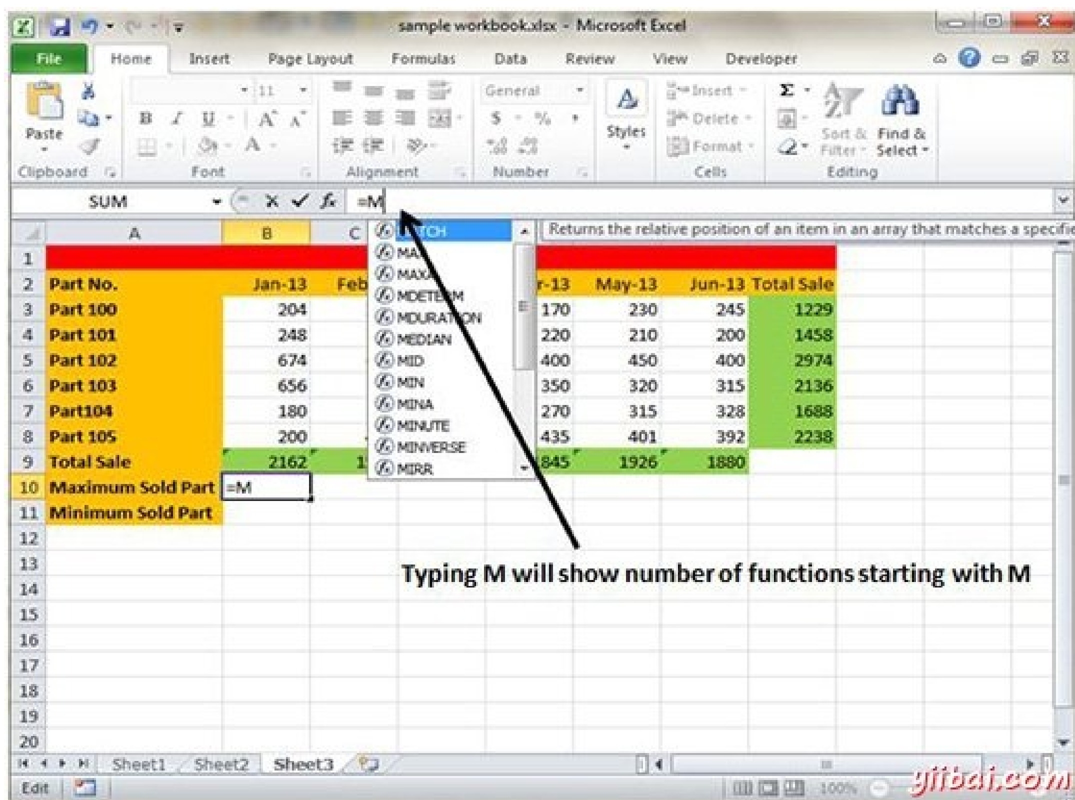
Excel使用函数 - Excel教程

在公式中的函数

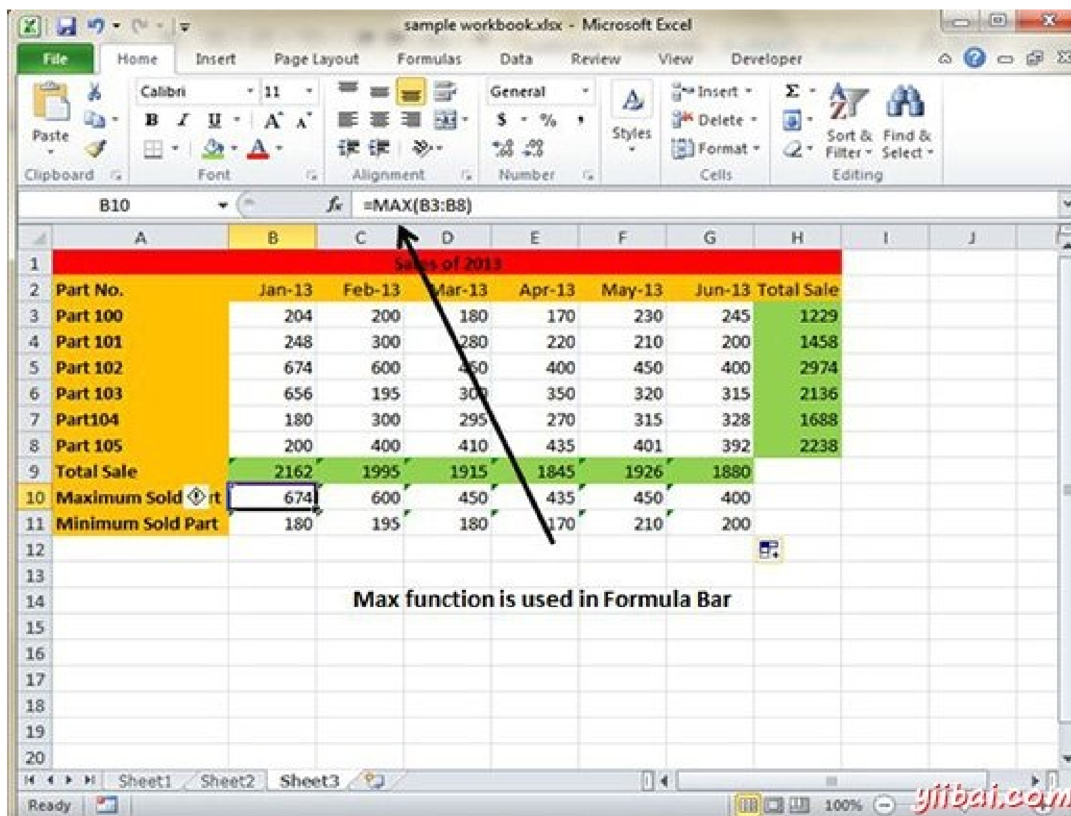
许多公式创建利用现有工作表函数。这些函数使您可以大大提升公式的能力和执行计算，如果使用运符有困难。例如，您可以使用日志或正弦函数来计算对数或Sin比例。 通过单独使用数学运算符，不能做复杂的计算。

使用函数

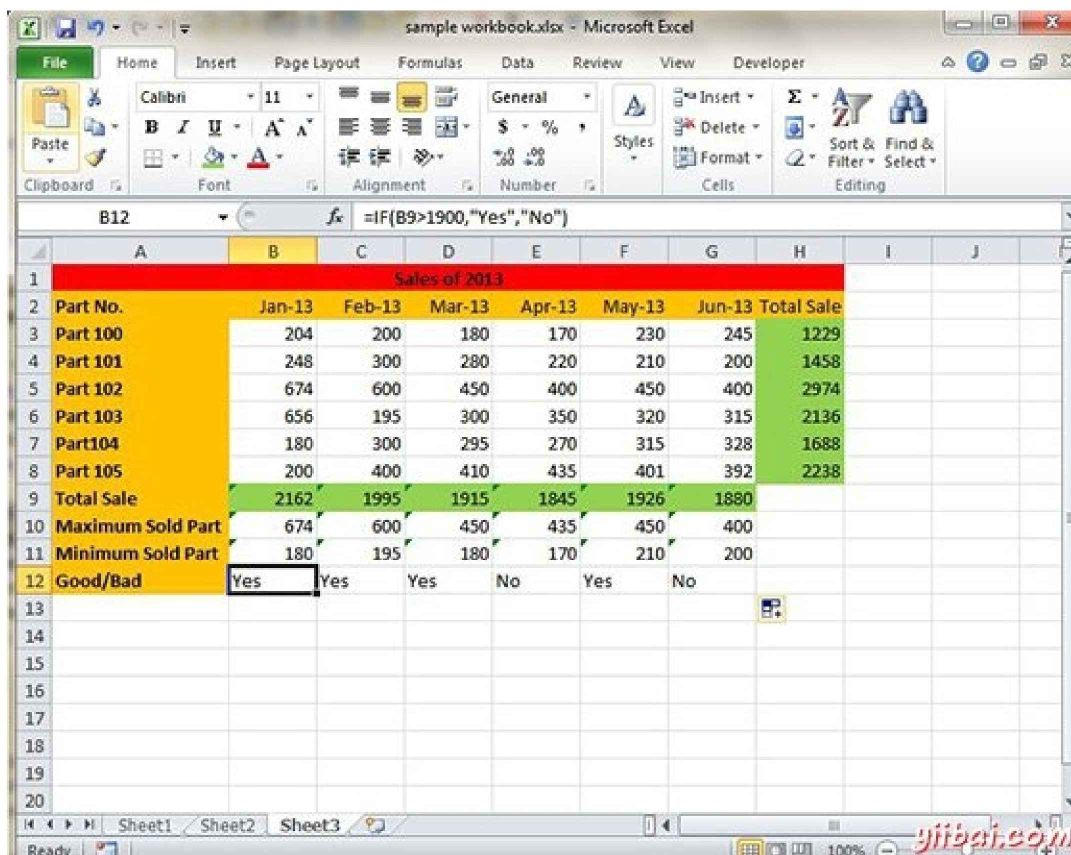
当键入等号(=)，然后键入任何字母，你会看到如下的搜索功能。



假设需要确定的范围内的最大值。公式不能告诉你不使用函数回答。 我们将用采用MAX函数式返回范围的最大值 B3:B8 as =MAX(A1:D100)



函数的另一个例子。假设你要找到，如果一个月的单元大于1900，那么我们可以给销售代表奖金。在我们可以编写公式IF函数为实现这一目标 =IF(B9>1900,"Yes","No")



函数的参数

在上面的例子中，你可能已经注意到，所有的函数中使用括号。括号内的信息参数列表。

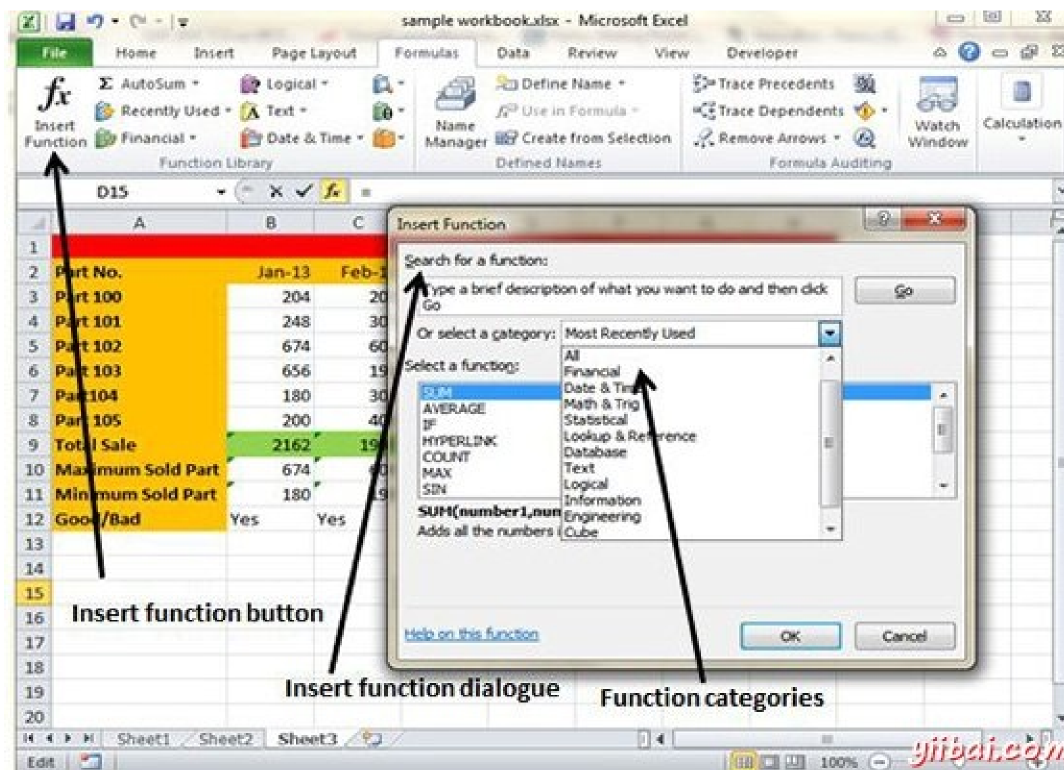
函数变化，它们如何使用不同的参数。根据它做什么，函数可以使用。

- 没有参数：例子: Now(),Date(),etc
- 一个参数: UPPER(),LOWER(),etc
- 参数固定数量：IF(),MAX(),MIN(),AVERGAGE(),etc.
- 无限数目的参数：
- 可选参数：

Excel内置函数 - Excel教程

内置函数

MS Excel中有许多内置的函数，我们可以在公式中使用。若要查看类别中的所有函数，选择公式选项卡»插入函数。然后插入函数对话框出现，从中我们可以选择的函数。



函数类别

让我们来看看一些在建在MS Excel的函数。

- 文本函数
    - UPPER : 所有字符转换成在提供的文本字符串为小写
    - UPPER : 所有字符转换成在提供的文本字符串为大写
    - TRIM : 删除重复的空间，并在一个文本串的开始和结束的位
    - CONCATENATE : 连接在一起的两个或多个文本字符串
    - LEFT : 返回指定的字符数，从所提供的文本字符串的开始
    - MID : 返回指定的字符数从提供的文本字符串的中间

- RIGHT : 返回指定的字符数从提供的文本字符串的结尾
- LEN : 返回一个提供的文本字符串的长度。
- FIND : 返回从提供的文本字符串中的字符提供或文本字符串的位置(区分大小写)

- Date & Time
  - DATE : 返回日期，从用户提供的年，月和日
  - TIME : 返回一个时间，从用户提供的时，分，秒
  - DATEVALUE : 表示日期，表示在Excel的日期 - 时间码的时间的文本字符串，转换为整数
  - TIMEVALUE : 表示一个时间，它表示在Excel中时间的文本字符串，十进制转换
  - NOW : 返回当前的日期和时间
  - TODAY : 返回当今的日期
- Statistical
  - MAX : 从提供的数据列表中返回最大值
  - MIN : 从提供的数值列表中返回最小值
  - AVERAGE : 返回提供的数值列表的平均
  - COUNT: 返回在供给组单元格或值的数值的个数
  - COUNTIF : 返回单元的数目(一个供给范围)，满足给定的标准
  - SUM : 返回供给的数字列表的总和
- Logical
  - AND : 测试了一些用户定义的条件，如果所有的条件计算为TRUE返回TRUE，否则返回FALSE。
  - OR : 测试了一些用户定义的条件，如果任何条件计算为TRUE返回TRUE，否则返回FALSE。
  - NOT : 返回一个逻辑值，为用户的提供相反的逻辑值或表达式，即返回FALSE是提供的参数为TRUE，如果提供的参数为FALSE返回TRUE)
- Math & Trig
  - ABS : 返回所提供的数字的绝对值(即模量)
  - SIGN : 返回一个供给数的符号sign(+ 1，-1或0)

- SQRT : 返回一个给定数的正平方根
- MOD : 返回两个提供的数字之间从一个除法的余数

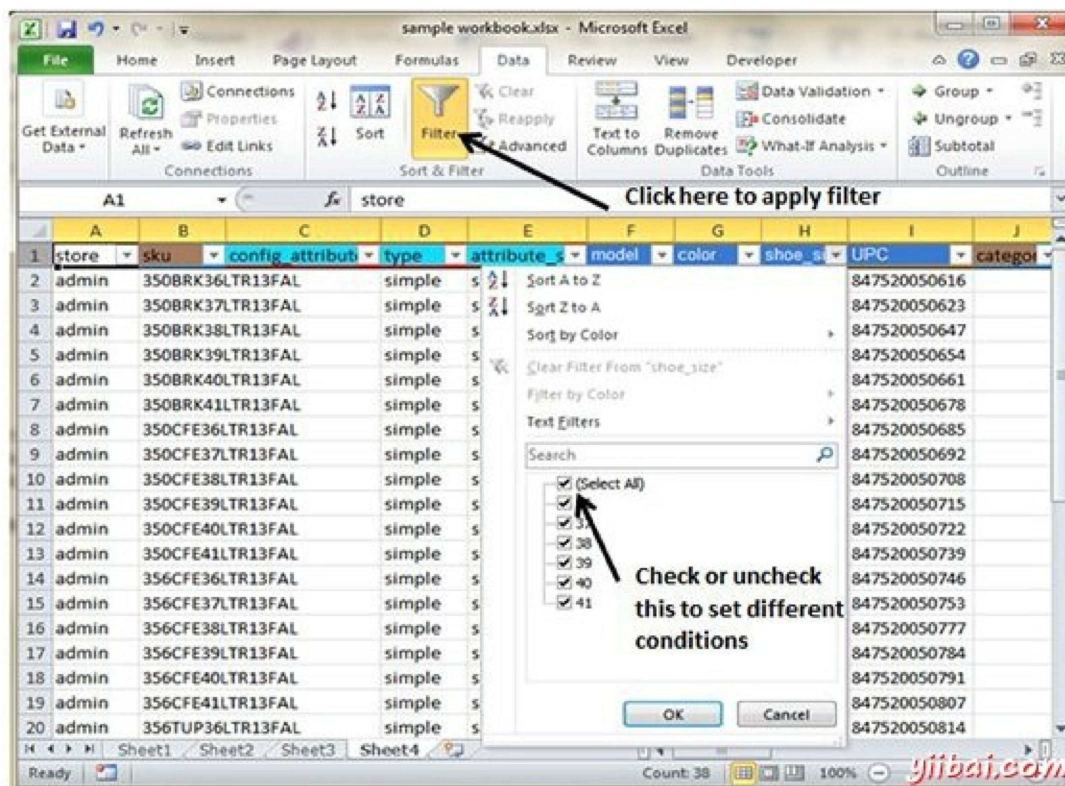
Excel数据过滤 - Excel教程

在MS Excel的过滤器

在MS Excel的数据筛选是指只显示符合某些条件的行(其他行被隐藏。)

使用存储数据，如果有兴趣希望看到其中鞋码为36的数据。然后，您可以设置过滤器来做到这一点。请按照以下步骤来做到这一点

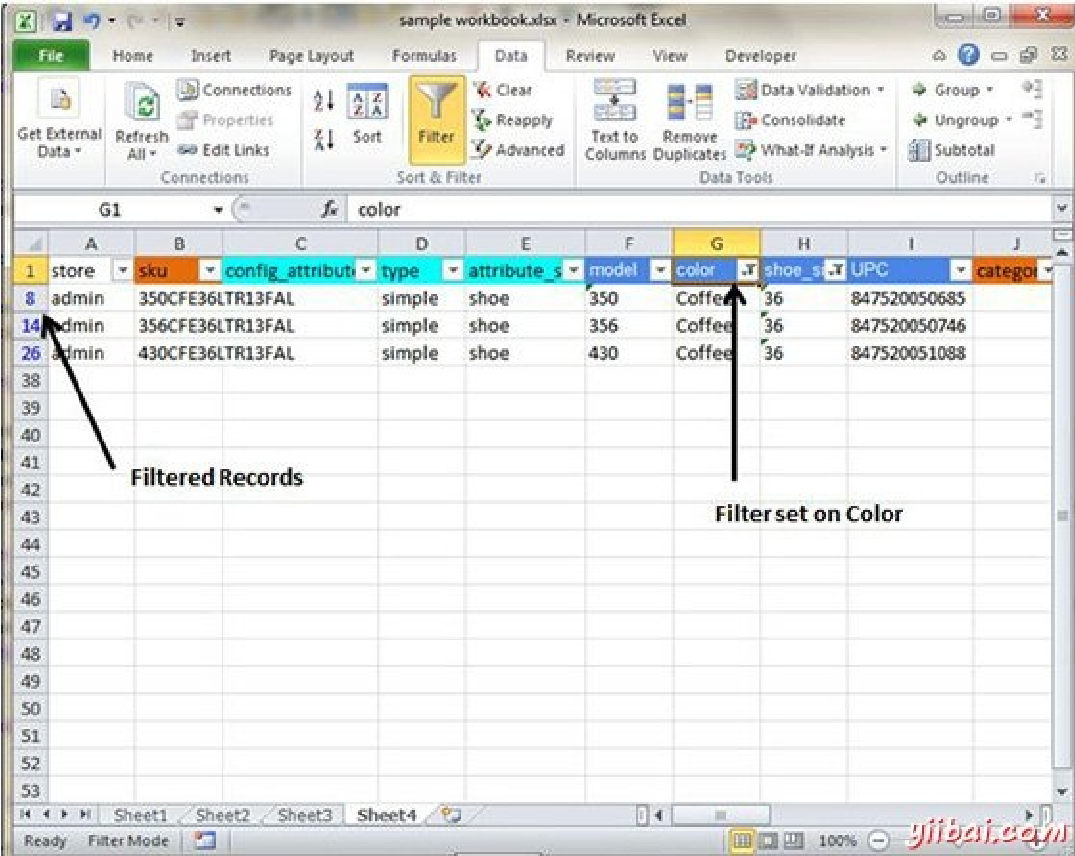
- 将光标置于标题行
- 选择数据选项卡»过滤器，设置过滤器



- 单击区域行标题的下拉箭头，并删除全部选中的对勾取消选择。
- 然后选中关口尺寸36，将过滤鞋码36的数据并显示数据
- 一些行的数据丢失; 这些行包含过滤(隐藏)数据。
- 有在该地区列下拉箭头，现在显示不同的图形 - 一个图标，指示列过滤

使用多个过滤器

可以通过即通过多列值的多个条件筛选记录。假设大小是36在过滤后，需要有过滤器，其中颜色是等于咖啡色。设置过滤器鞋码后，选择颜色列，然后设置过滤器颜色


Filtered Records
Filter set on Color

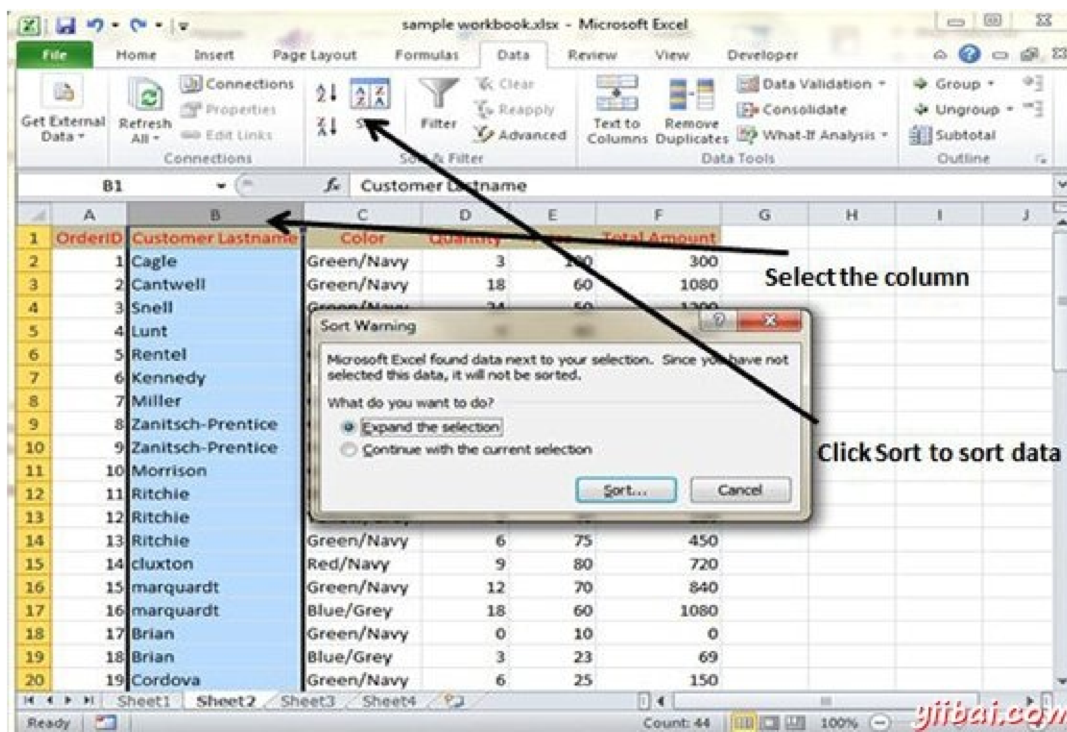
Excel数据排序 - Excel教程

在MS Excel排序

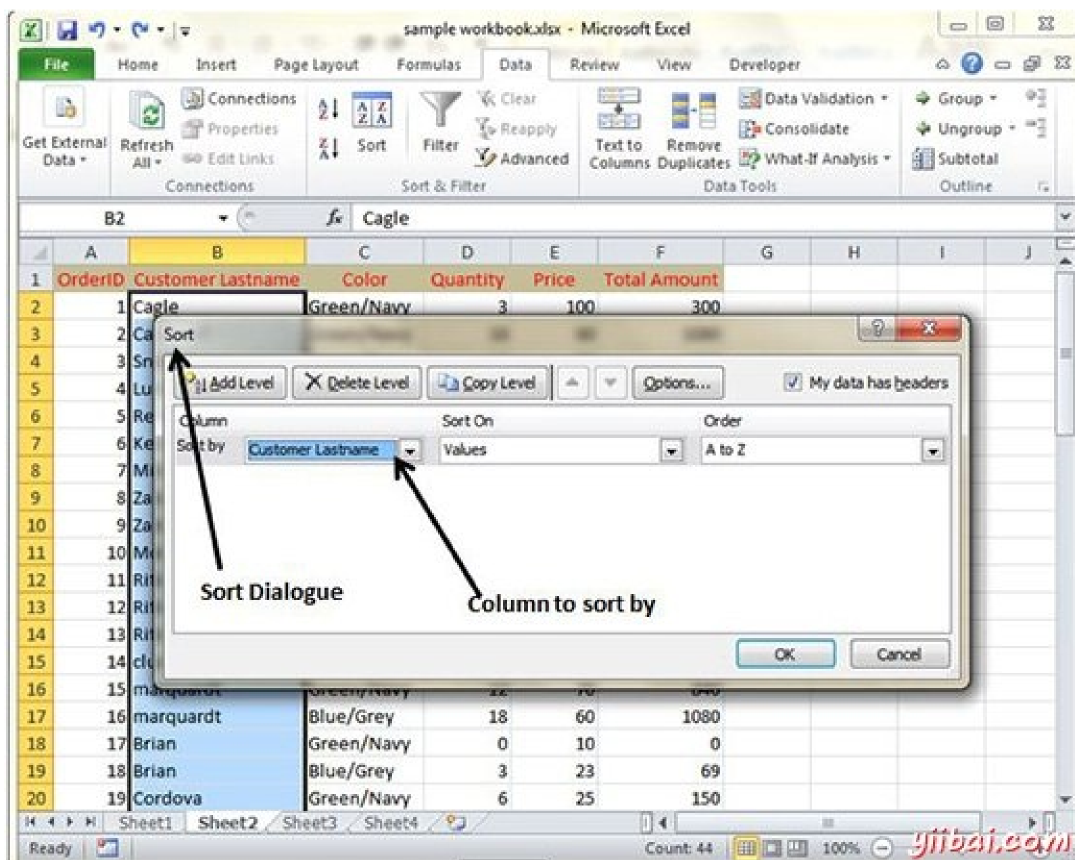
在MS Excel数据重新排列基于特定列的内容的行上。可能要排序表把名字的字母顺序。或者，也许想通过数据量从最小到最大或最大到最小排序。

排序的数据按照以下步骤。

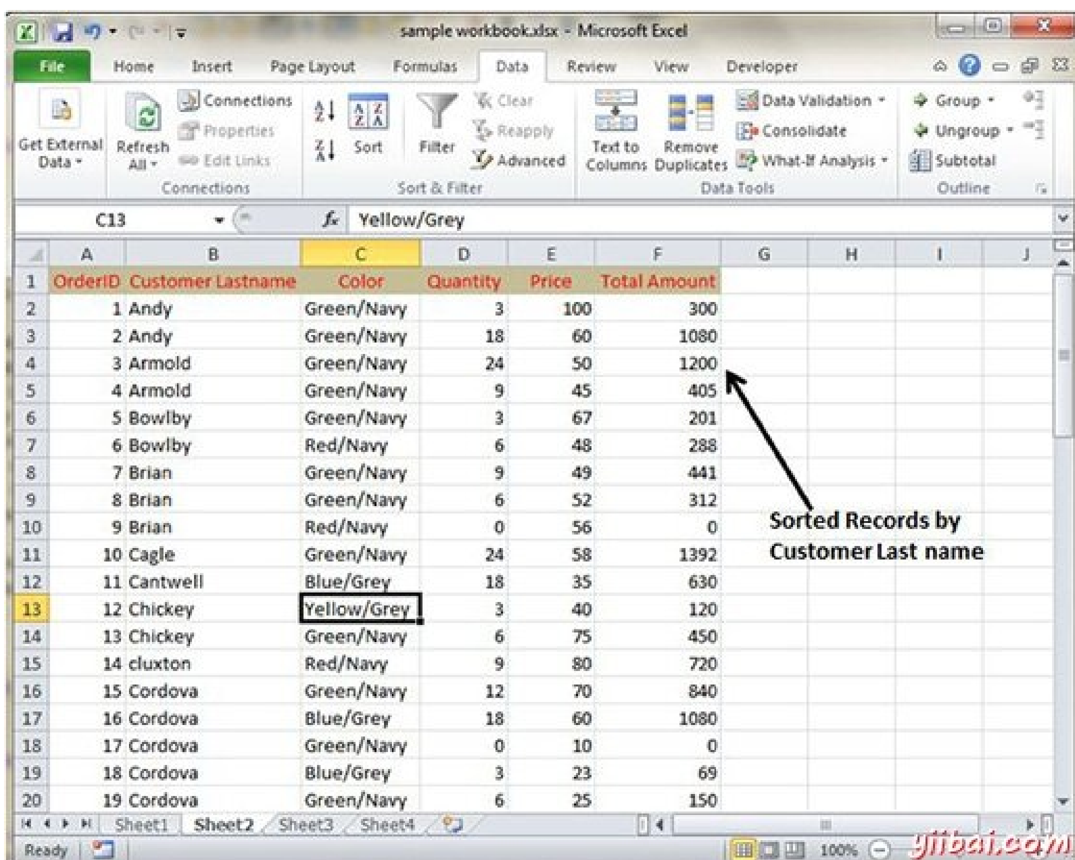
- 通过选择想要的数据排序的列。
- 选择数据选项卡»排序，出现以下对话框



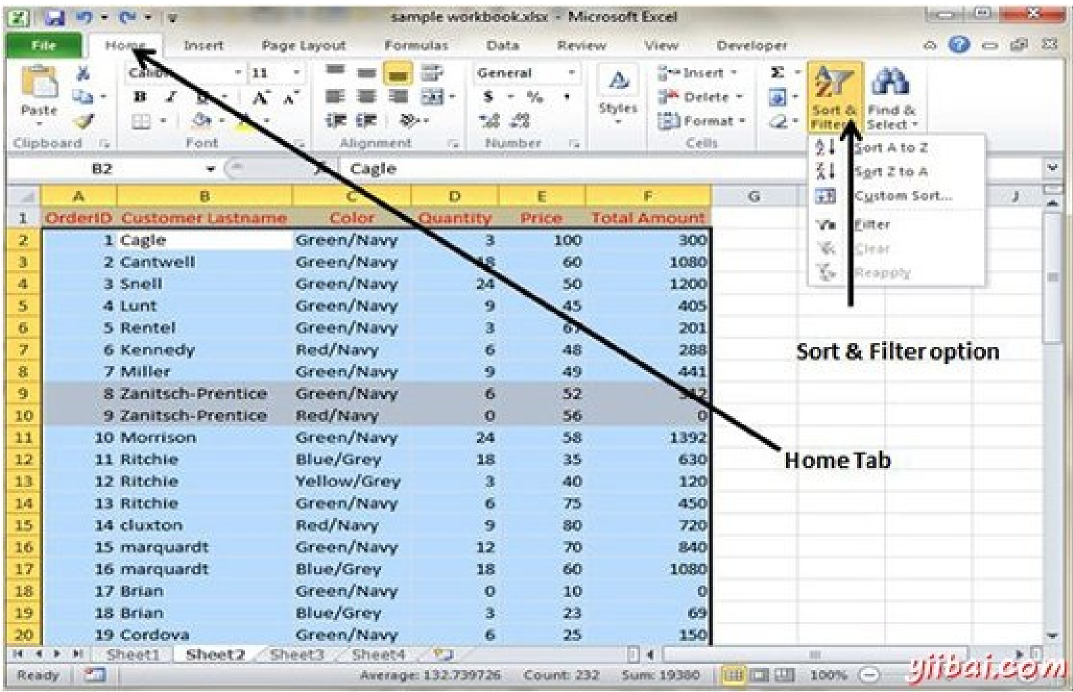
- 如果你想进行排序的基础上选定的列数据选择继续使用选择，或如果想基于其他列排序选择展开选择。
- 您可以排序基于以下条件。
  - 值：字母或数字
  - 单元格颜色：基于单元的颜色
  - 字体颜色：根据字体颜色
  - 单元格图标：基于单元格图标



- 单击确定将数据进行排序。



排序选项也可以从主页选项卡。选择主页选项卡»排序和筛选。可以看到相同的对话框排序记录。


Sort & Filter option
Home Tab

Excel使用范围 - Excel教程

使用范围

单元是在工作表中，可容纳一个值，一些文字或公式的单个元素。一个单元是由它们的地址，其中包括其列字母和行号的。例如，单元格B1是在第二列和第一行的单元格。

一组单元被称为一个范围。指定的地址范围内通过指定的左上角单元格地址和右下角单元格地址，用冒号分隔

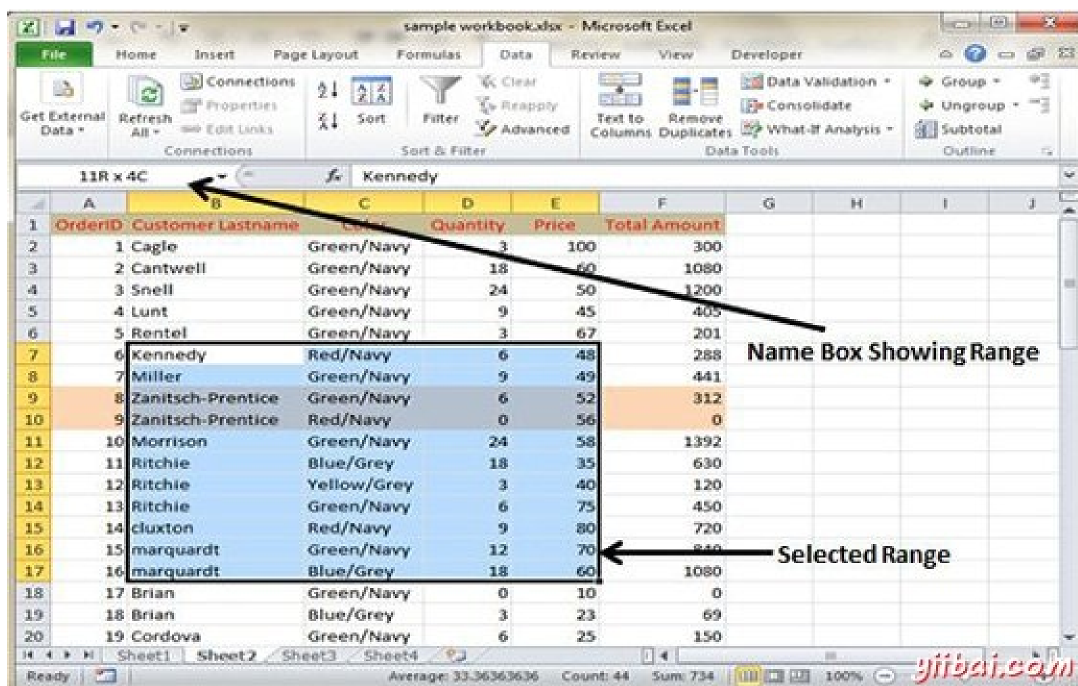
范围的例子

- **C24 :** 由单个单元的范围
- **A1:B1 :** 两个单元格占据一行和两列
- **A1:A100 :** 100个单元在A列
- **A1:D4 :** 16单元格(四行四列)

选择范围

可以选择的范围在几个方面：

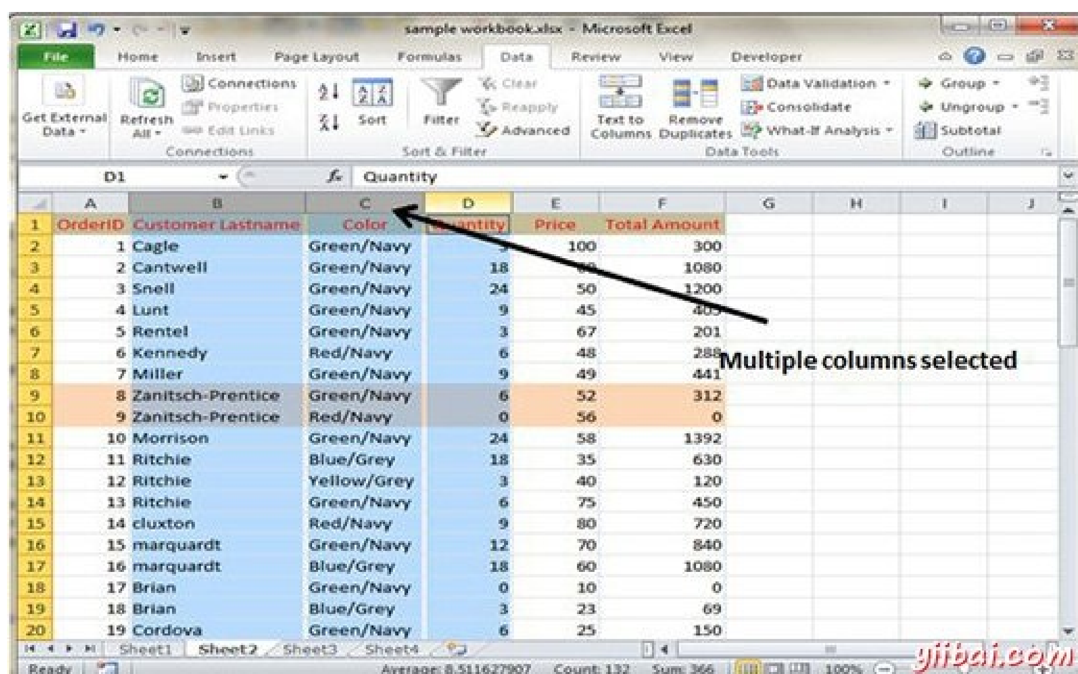
- 按下鼠标左键并拖动，突出范围。 然后松开鼠标按钮。如果拖动到屏幕的结束，工作表将滚动。
- 按住Shift键的同时使用导航键选择一个范围。
- 按F8键，然后移动单元格指针与导航键突出显示范围。按F8再次向导航键恢复正常运行。
- 键入单元格或单元格区域的地址到名称框，然后按Enter键。 Excel中选择您指定的单元格或区域。



选择完整的行和列

当你需要选择一个整行或列。可以选择为选择范围整个行和列在大致相同的方式：

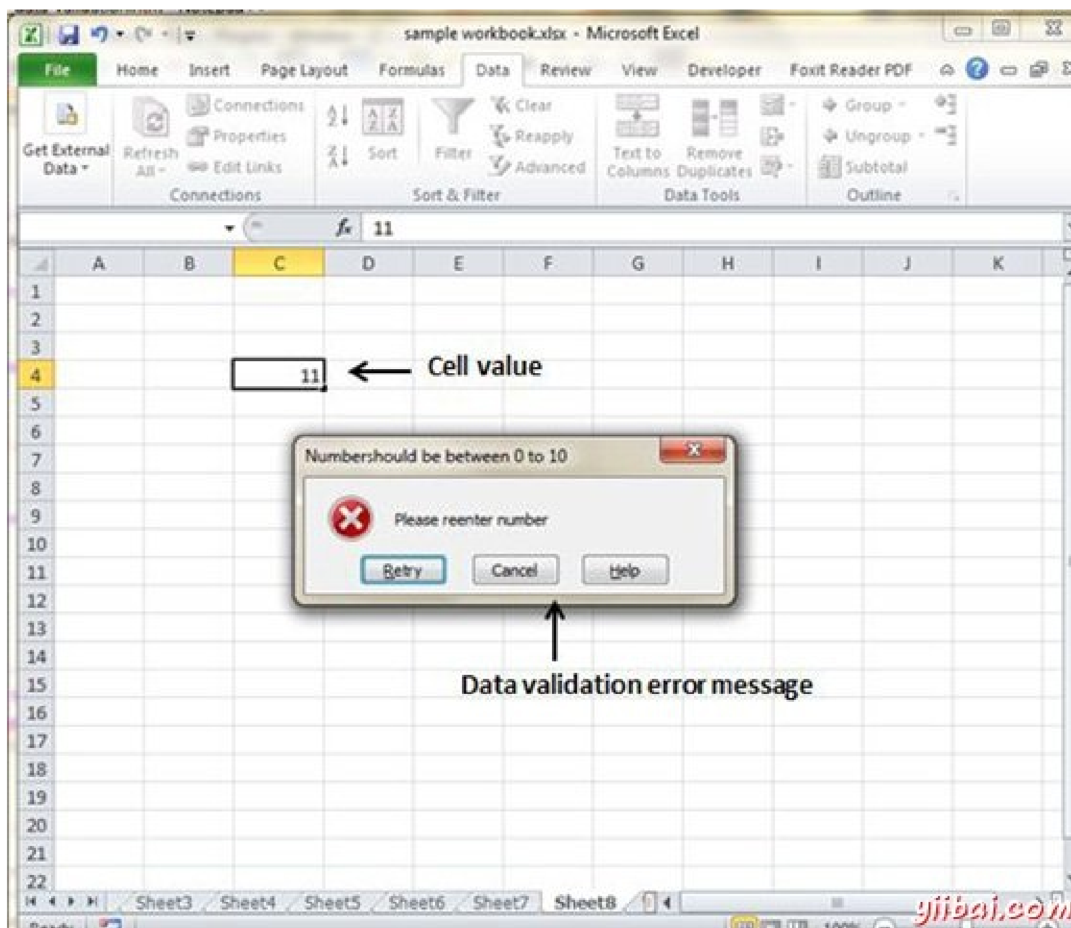
- 单击行或列边界，以选择单个行或列。
- 要选择多个相邻的行或列，单击行或列边框并拖动突出额外的行或列。
- 要选择多个（不相邻的）行或列，按Ctrl键的同时单击所需的行或列边框。



Excel数据验证 - Excel教程

Excel数据验证:

MS Excel数据验证功能，您可以设置某些规则决定什么可以输入到单元格。 例如，您可能希望单元格限制在0到10之间的特定整数数据录入。如果用户进行无效输入，就可以显示自定义消息，如下图所示



验证标准

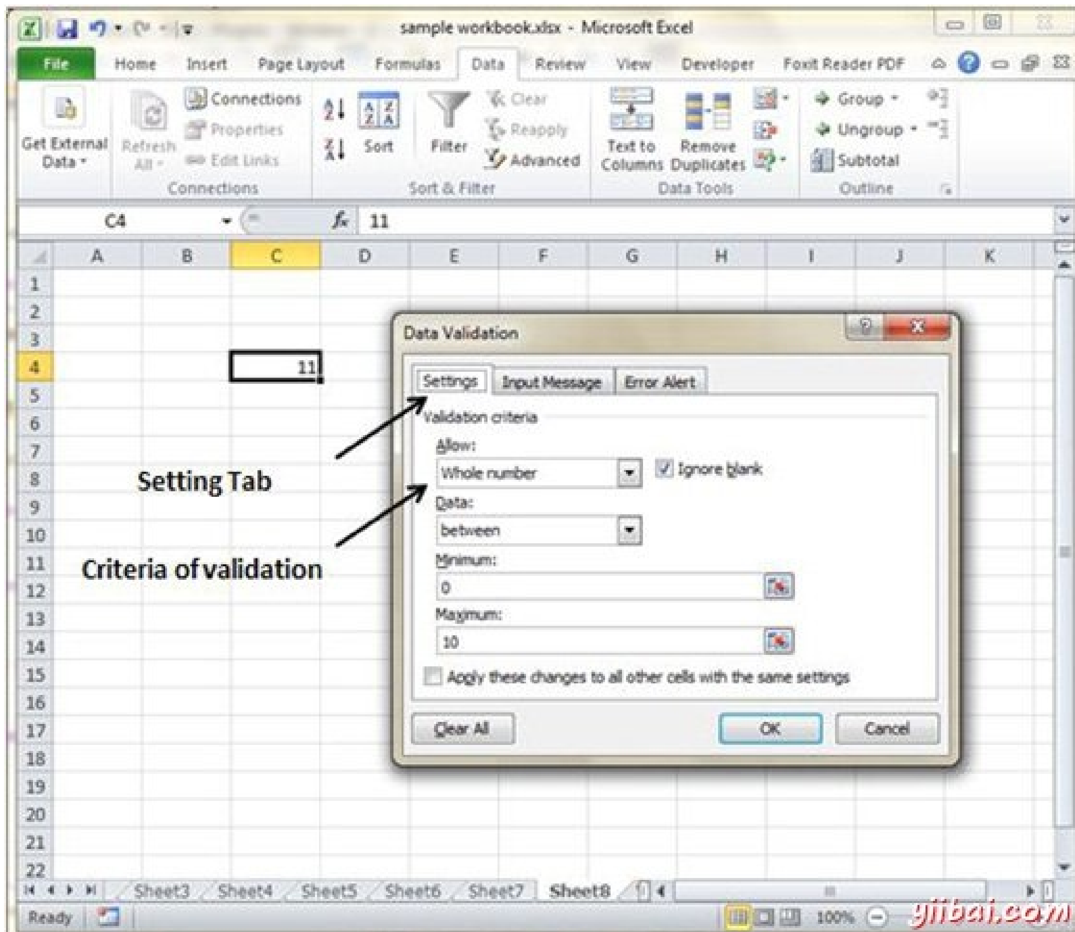
要指定允许数据在一个单元格或单元格区域的类型，请在您参考其中显示了数据验证对话框的所有三个选项卡下面的步骤。

- 选择单元格或单元格区域。
- 选择数据»数据工具»数据验证。 Excel会显示有3个选项卡设置，输入信息和错误警报的数据验证对话框。

设置选项卡

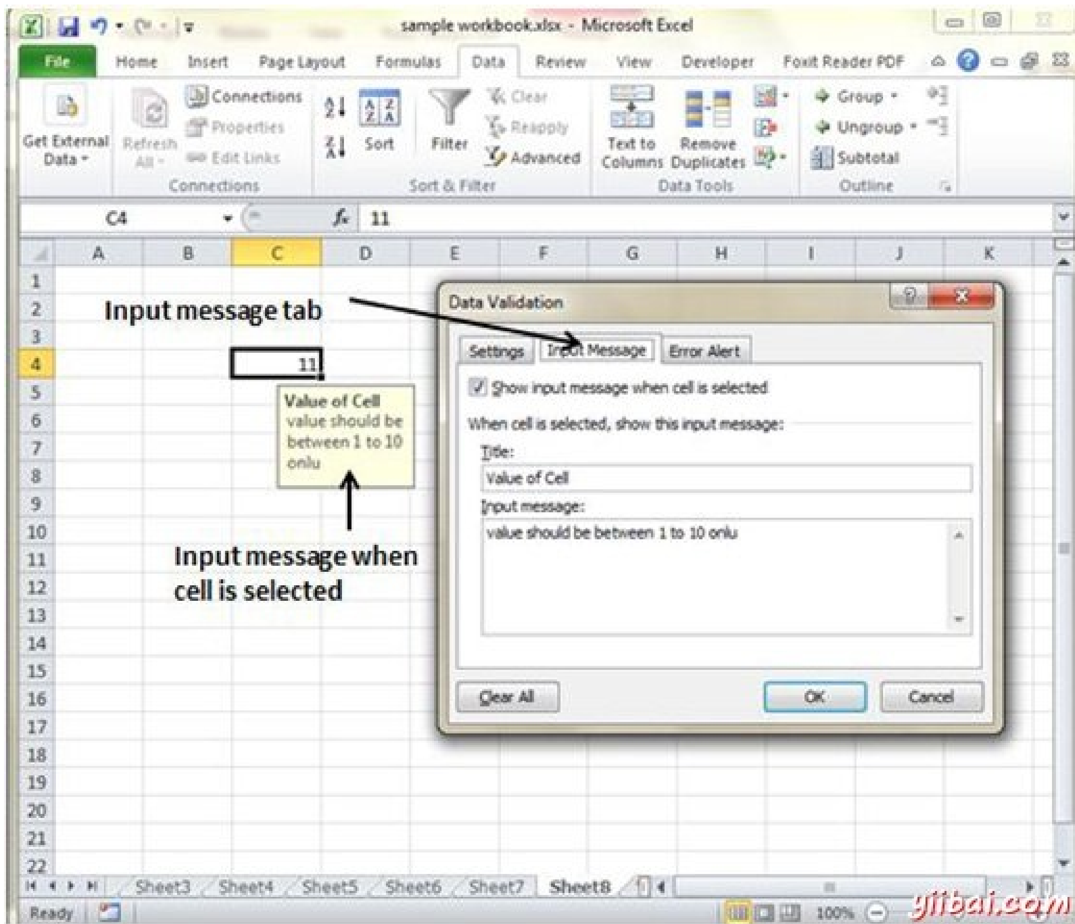
在这里，您可以设置验证您所需要的类型。选择从允许下拉列表中选择一个选项。数据验证对话框的内容将发生变化，显示基于您选择的控制。

- 任何值：选择此选项删除任何现有的数据验证。
- 整数：用户必须输入一个整数。例如，您可以指定该条目必须大于或等于50。
- 十进制：用户必须输入一个数字。例如，可以指定该条目必须大于或等于10且小于或等于20。
- 列表：用户必须从你提供的条目列表中选择。创建下拉列表使用此验证。必须给输入范围，然后这些值将出现在下拉。
- 日期：用户必须输入一个日期。你从数据下拉列表中选择指定一个有效日期范围。例如，可以指定输入的数据必须大于或等于1月1日到2013年，且小于或等于2013年12月31日。
- 时间：用户必须输入时间。从数据下拉列表中选择指定一个有效的时间范围。例如，可以指定输入的数据必须晚于12:00 PM
- 文本长度：数据的长度(字符数)是有限的。通过使用数据下拉列表中指定一个有效的长度。例如，可以指定输入的数据的长度为1(单一字母数字字符)。
- 自定义：要使用此选项，则必须提供一个逻辑公式，确定用户的条目(逻辑公式返回TRUE或FALSE)的有效性。



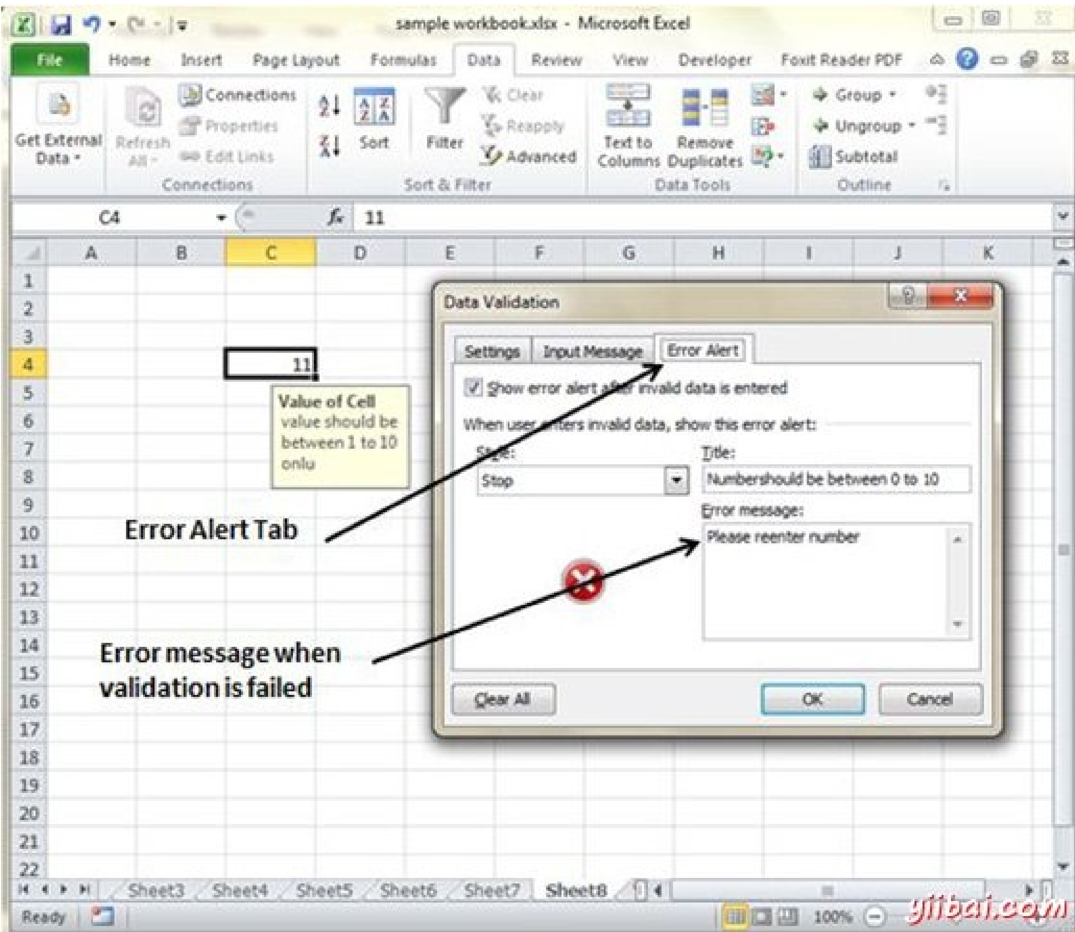
输入信息选项卡

可以设置输入帮助消息此选项卡。填充输入消息选项卡的标题和输入信息和输入信息时被选中单元格会出现..



错误通知选项卡

可以使用此选项卡指定错误消息。填写标题和错误信息。选择错误的停止，警告或信息按照你需要的风格。


sample workbook.xlsx - Microsoft Excel
Data Validation
Settings
Input Message
Error Alert
Show error alert after invalid data is entered
When user enters invalid data, show this error alert:
Style:
Stop
Title:
Numbershould be between 0 to 10
Error message:
Please reenter number
Clear All
OK
Cancel
Value of Cell
value should be between 1 to 10 only
Error Alert Tab
Error message when validation is failed

Excel使用样式 - Excel教程

在MS Excel中使用样式

使用MS Excel命名风格使它很易于应用一组预定义的格式设置选项的单元格或单元格区域。这样可以节省时间以及确保单元格一致的外观。

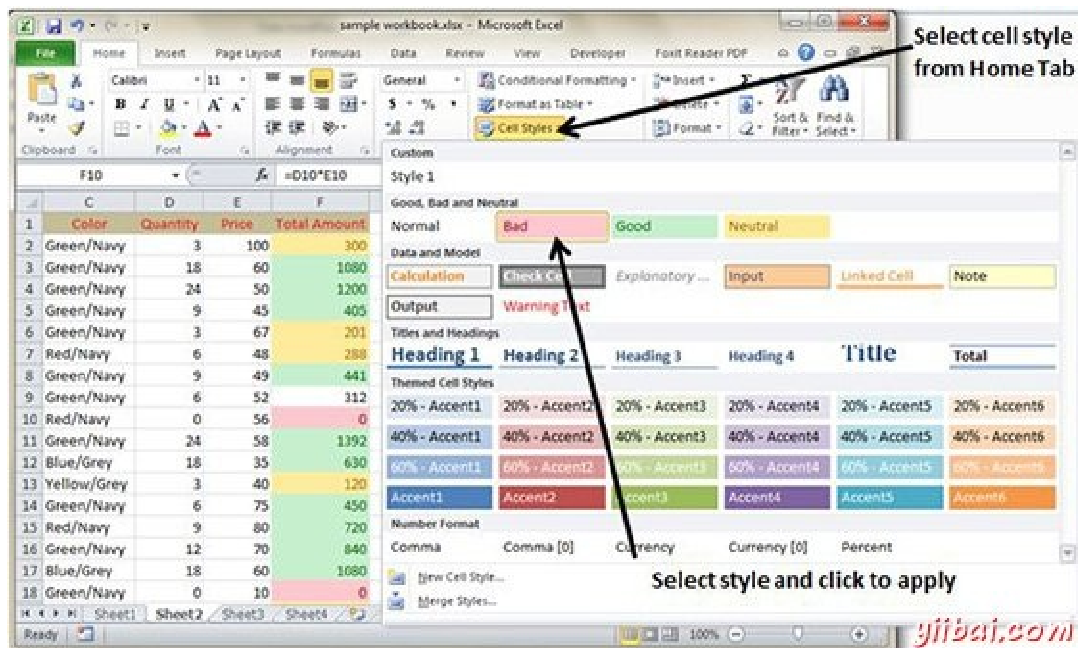
风格可以设置多达六种不同的属性：

- 数字格式
- 字体(类型，大小和颜色)
- 对齐(垂直和水平)
- 边框
- 模式
- 保护(锁定和隐藏)

现在，让我们来看看样式如何是有帮助的。假设应用一个特定的风格，以二十几个单元格散在整个工作表。后来，你会发现这些单元应该是12PT而不是14PT的字体大小。 不是改变每个单元，只需编辑样式。所有单元格与特定的样式就会自动改变。

应用样式

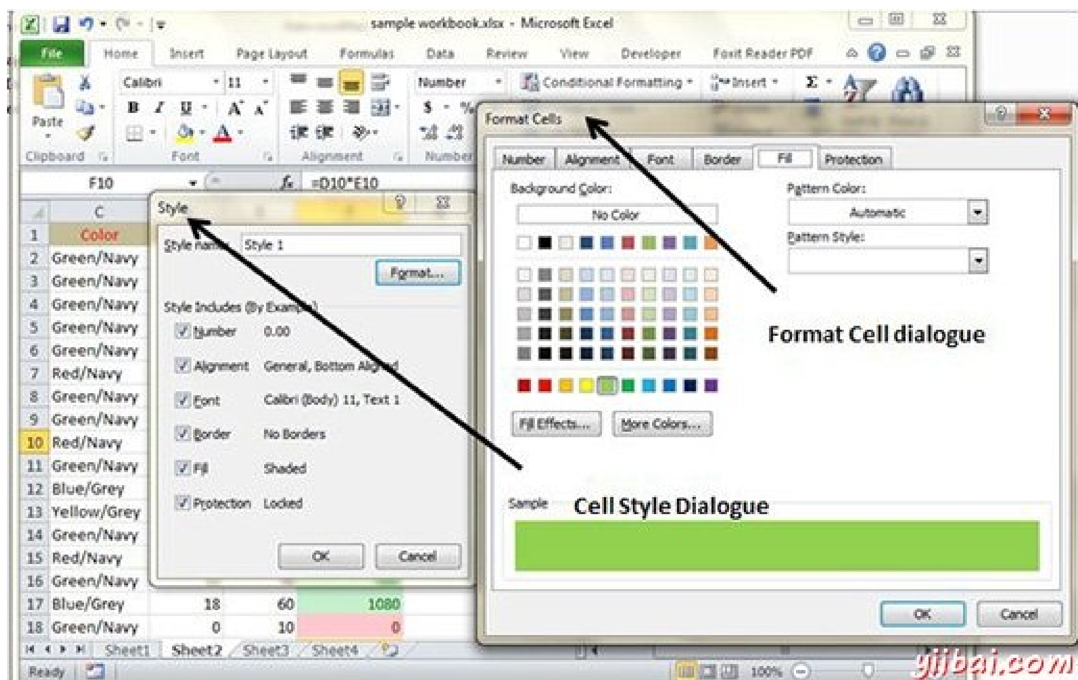
选择主页»样式»单元格样式。请注意，这显示是实时预览：那就是，当你将鼠标移到风格选择，选定单元格或区域暂时显示样式。当你看到喜欢的样式，单击它的样式应用到选择。



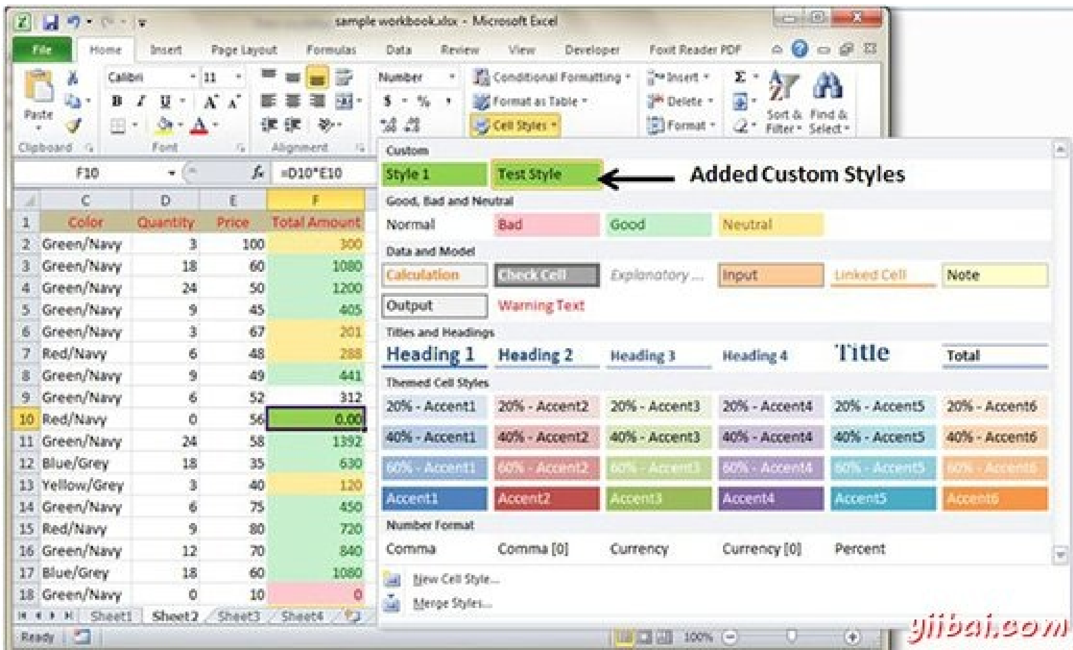
在MS Excel中创建自定义样式

我们可以在Excel 中创建新的自定义样式，用来创建一个新的样式，请按照下列步骤操作：

- 选择一个单元格，然后单击从主页选项卡单元格样式
- 单击新建单元格样式，并命名样式名称
- 单击格式应用格式的单元格



- 经过应用的格式，点击确定。这将在款式增添新的样式。您可以在首页»样式查看。


sample workbook.xlsx - Microsoft Excel
Custom
Style 1
Test Style
Added Custom Styles
Good, Bad and Neutral
Normal
Bad
Good
Neutral
Data and Model
Calculation
Check Cell
Explanatory ...
Input
Linked Cell
Note
Output
Warning Text
Titles and Headings
Heading 1
Heading 2
Heading 3
Heading 4
Title
Total
Themed Cell Styles
Number Format
Comma
Comma [0]
Currency
Currency [0]
Percent
New Cell Style...
Merge Styles...
yiibai.com

Excel使用主题 - Excel教程

在MS Excel使用主题

为了帮助用户创造出更多具有专业外观的文档MS Excel中已纳入被称为文档主题的概念。使用主题是一种易于以指定的颜色，字体，和各种在文档的图形效果。最重要的是，改变文档的整体外观是一件轻而易举的事。点击几下鼠标是所有需要应用不同的主题，并改变你的工作簿的外观。

应用主题

选择页面布局选项卡»主题下拉。请注意，这显示是实时预览：那就是，当你在主题暂时显示效果主题，移动鼠标。当你看到你喜欢的样式，单击它的样式应用到选择。



在MS Excel创建自定义主题

我们可以在Excel 中创建新的自定义主题，以创建一个新的样式，请按照下列步骤操作：

- 点击保存在页面布局选项卡下的主题当前主题选项
- 这将保存当前主题到office文件夹。
- 您可以在以后浏览主题加载主题。

Excel使用模板 - Excel教程

在MS Excel中使用模板

模板本质上是一个模型，模型作为基础的东西。Excel模板是我们用来创建其他工作簿的工作簿。

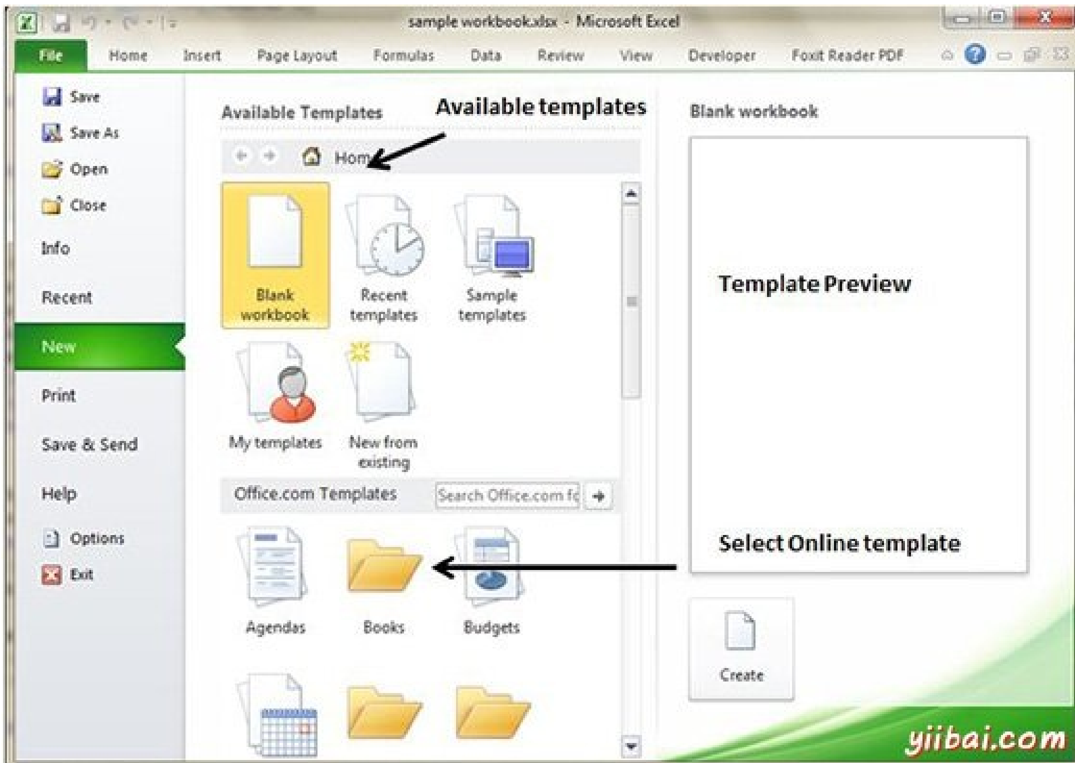
查看可用模板

要查看Excel的模板，选择文件»新建以显示Backstage视图可用模板画面。您可以选择存储在硬盘上的模板，或者从Microsoft Office Online模板。如果您选择从Microsoft Office Online模板，必须连接到Internet下载。Office 在线模板部分包含许多图标，其中表示各种类别的模板。点击一个图标，你会看到可用的模板。当您选择一个模板缩略图，可以看到在右侧面板中的预览。



在线模板

这些模板数据是Microsoft服务器在线提供的。当您选择模板，点击它。它会下载从微软服务器的模板数据和打开它，如下所示。

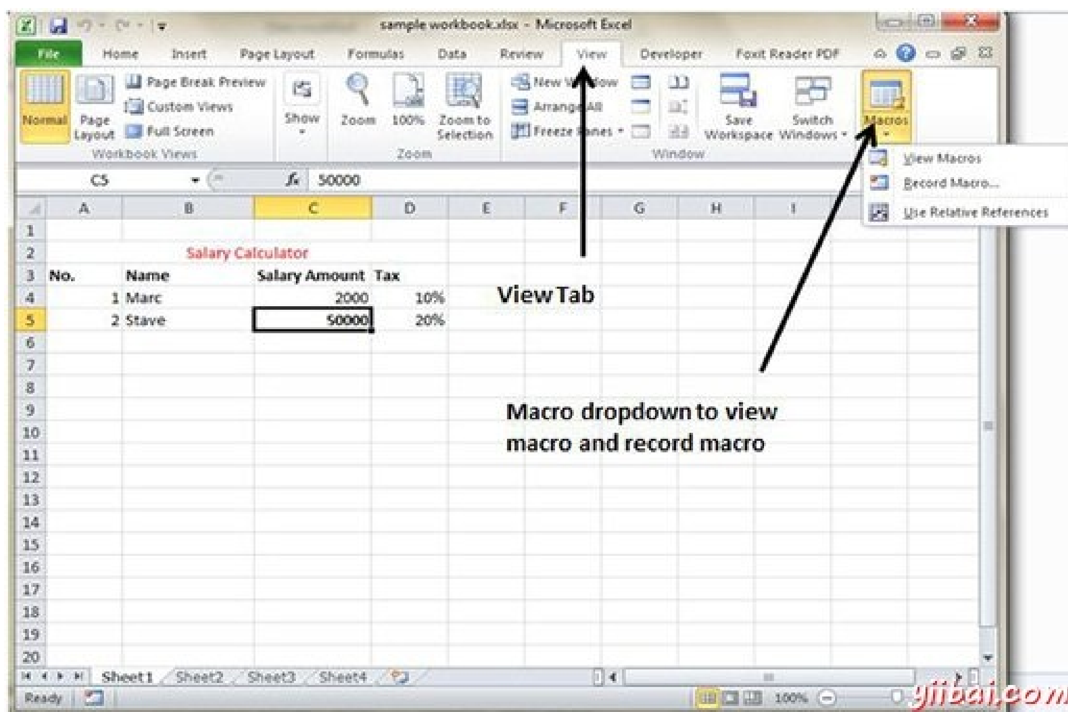

sample workbook.xlsx - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Foxit Reader PDF
Save
Save As
Open
Close
Info
Recent
New
Print
Save & Send
Help
Options
Exit
Available Templates
Available templates
Blank workbook
Recent templates
Sample templates
My templates
New from existing
Office.com Templates
Agendas
Books
Budgets
Blank workbook
Template Preview
Select Online template
Create
yiibai.com

Excel使用宏 - Excel教程

MS Excel宏

宏使您能够自动化，可以在Excel 2010中几乎承担所有的任务。通过使用视图选项卡的宏录制»宏，下拉来记录执行例行任务，不仅大大加快了程序，在任务的每一步是每一位执行任务的时候进行了同样的方式。

要查看宏选择查看标签»宏下拉



宏选项

查看选项卡包含一个宏命令按钮其中下拉菜单包含以下三个选项。

- 查看宏：打开宏对话框，您可以选择一个宏运行或编辑。
- 录制宏：打开，新的宏定义设置，然后启动宏录制录制宏对话框; 点击状态栏上的录制宏按钮。
- 使用相对引用：录制宏时使用相对单元格地址，使您能够在工作表比原来用在宏的录制以外的其它地区运行的宏。

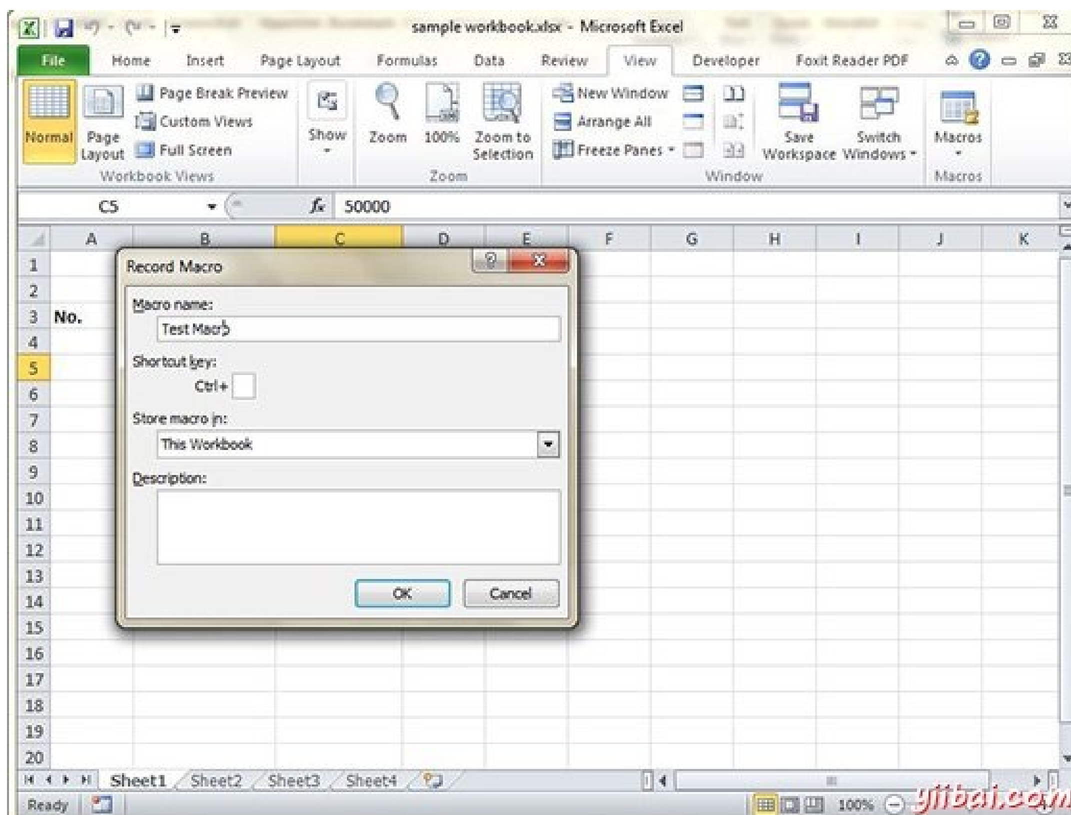
创建宏

可以使用两种方式创建宏：

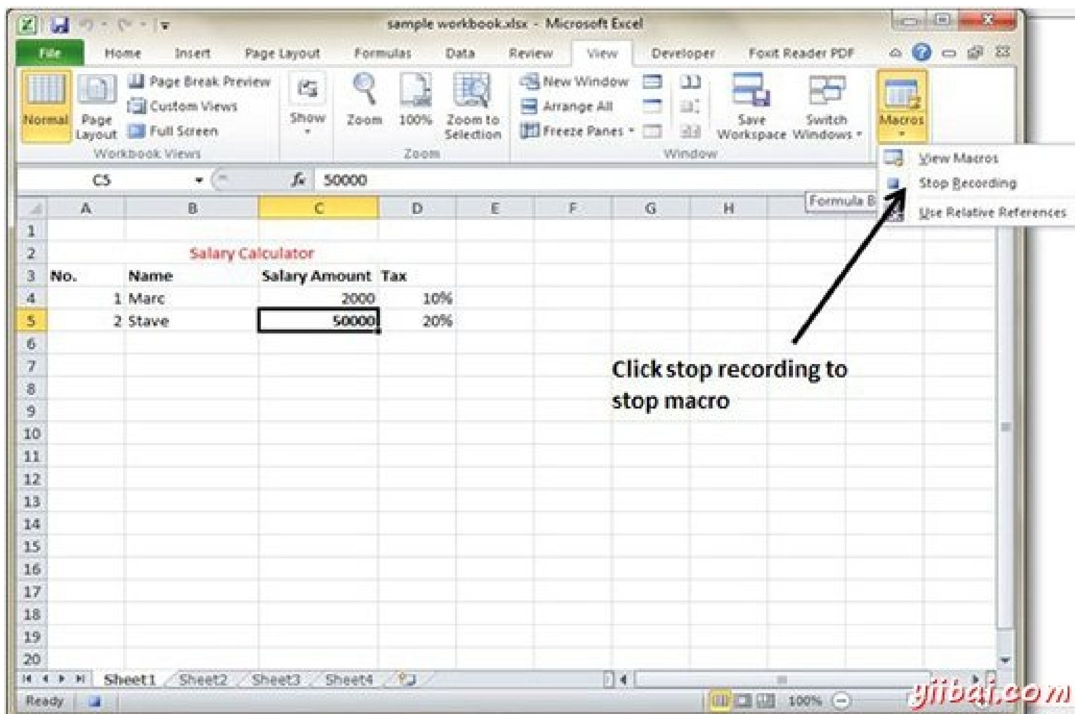
- 使用MS Excel的宏记录器来记录动作，在工作表进行
- 输入要遵循VBA代码，Visual Basic编辑器的说明

现在，让我们创建一个简单的简单的宏，将自动使单元格内容为粗体的任务和应用单元格颜色。

- 选择查看标签»宏下拉
- 点击录制宏如下

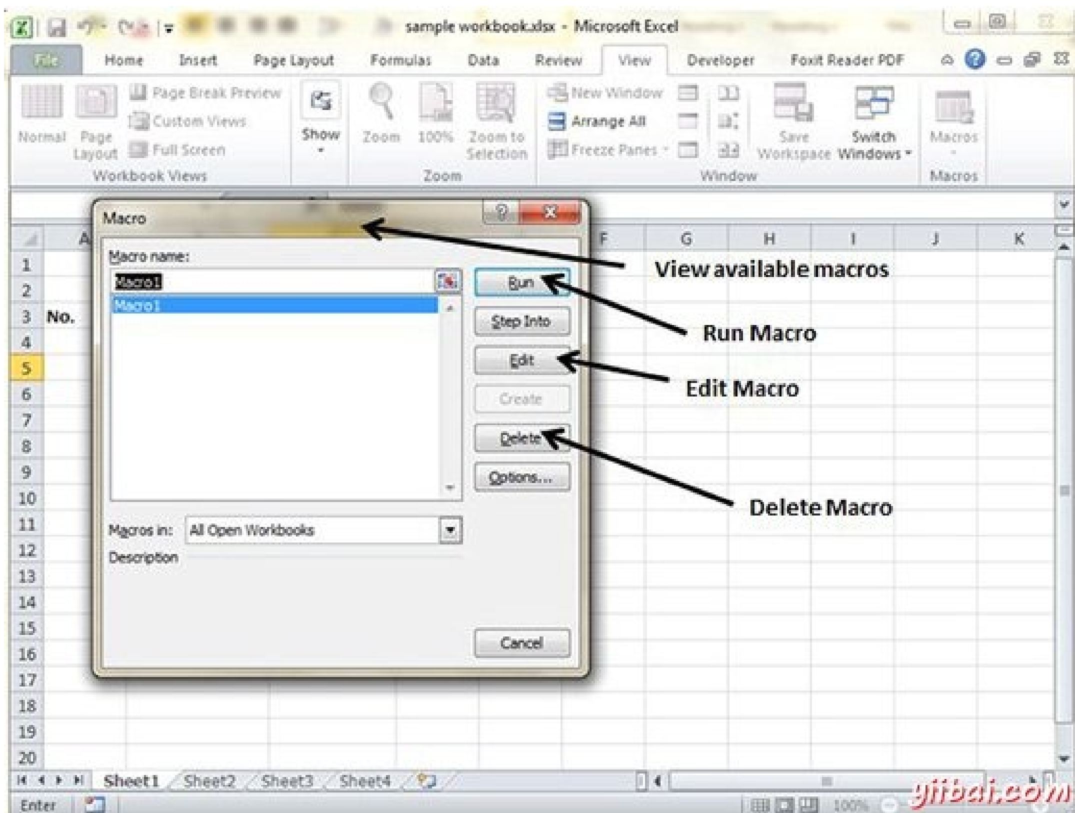


- 现在，宏将开始录制。
- 做动作要重复执行的步骤。宏会记录这些步骤。
- 一旦与所有步骤完成宏录制，可以停止。



编辑宏

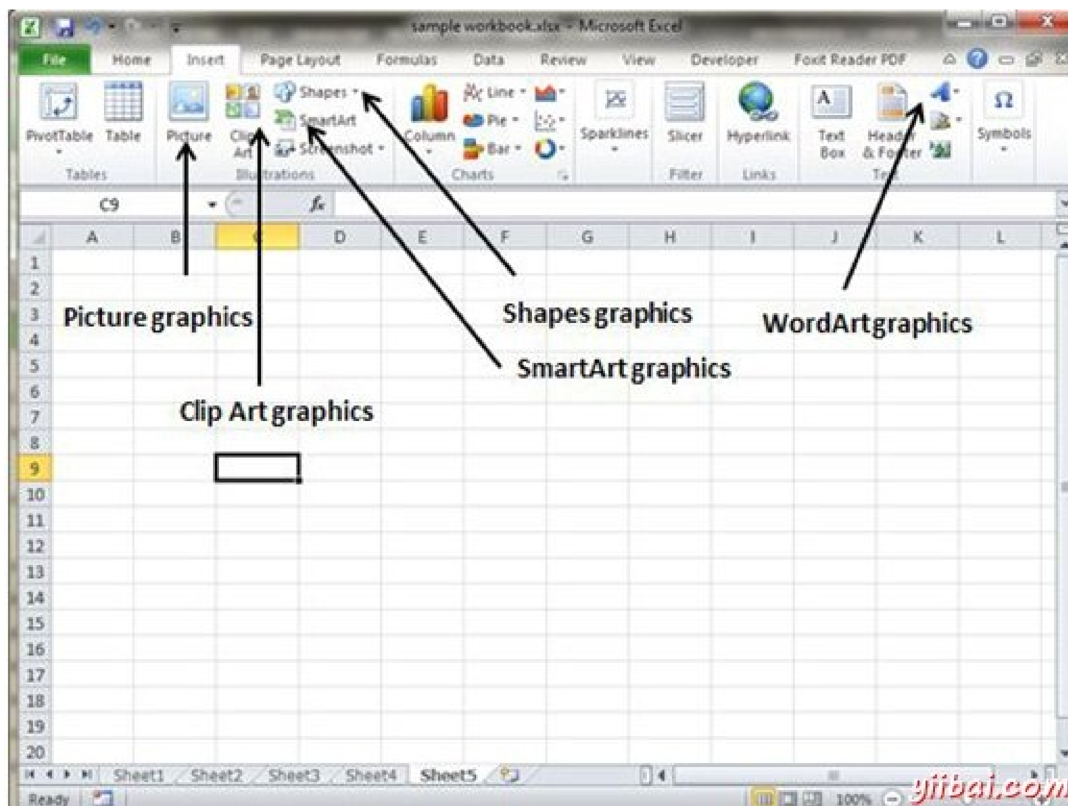
可以随时编辑创建宏。编辑宏将带你到VBA编程编辑器



Excel添加图形 - Excel教程

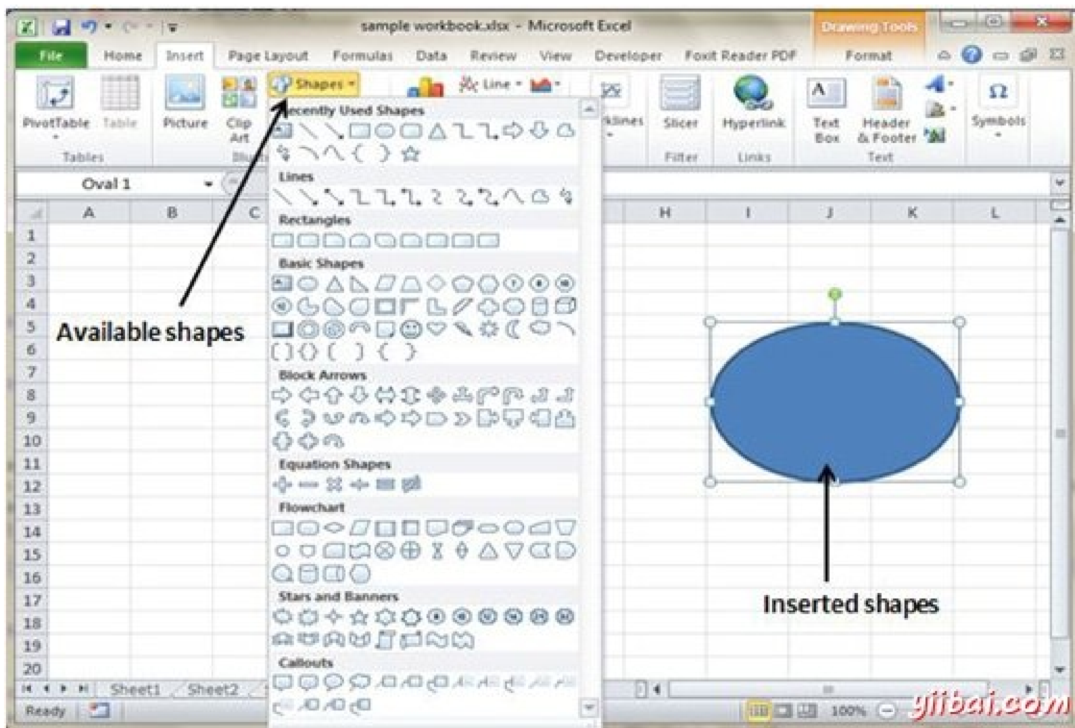
在MS Excel的图形对象

MS Excel支持各种类型的图形对象，如形状画廊，SmartArt，文本框和艺术字可用功能区在插入选项卡上。图形在插入选项卡可用。请参见下面的截图在MS Excel 2010中的各种可用的图形。



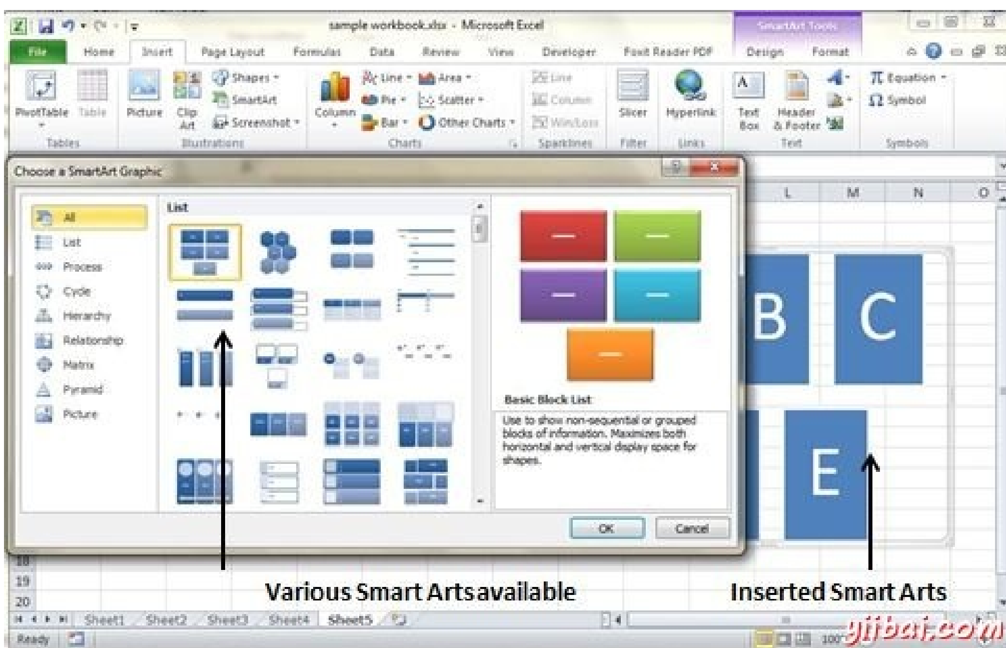
插入形状

- 选择插入选项卡»图形下拉
- 选择要插入的形状。单击形状将其插入。
- 编辑插入的形状只需拖动形状的鼠标。形状将调整形状。



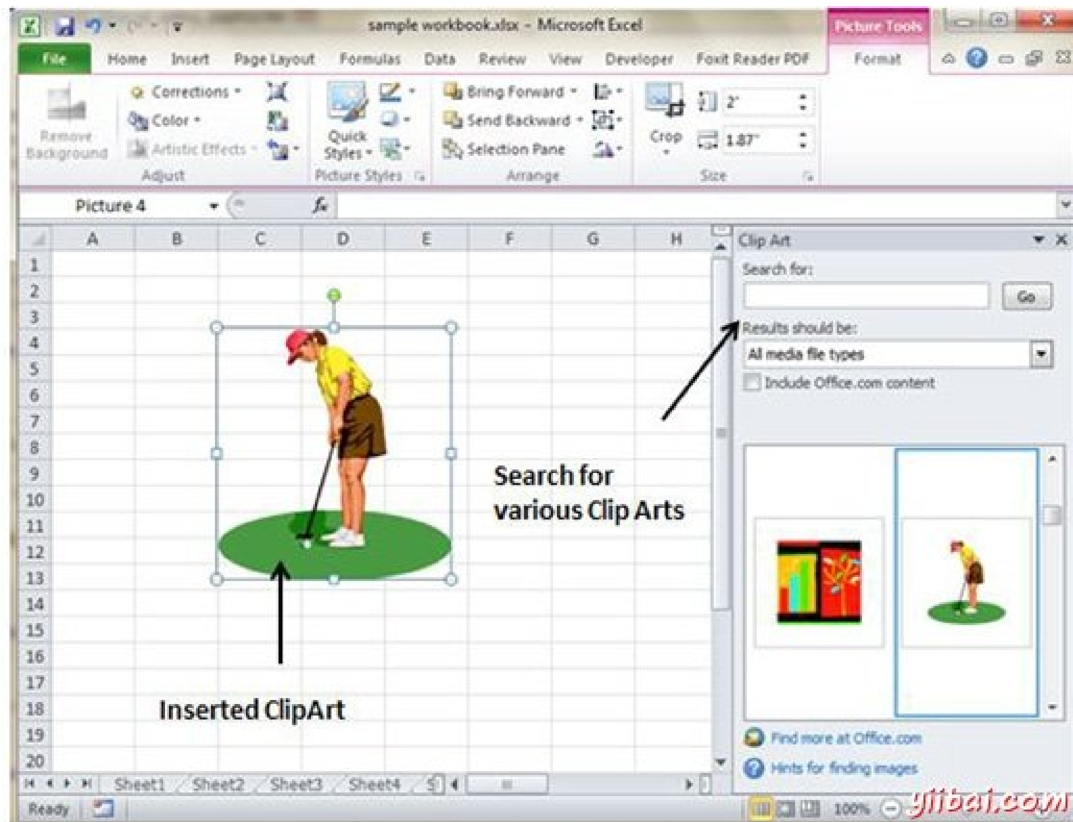
插入智能艺术

- 选择插入标签»SmartArt
- 点击SmartArt将打开SmartArt对话，如下屏幕截图。从可用smartArts列表选择
- 点击SmartArt将其插入工作表
- 编辑SmartArt，按你的需要



插入剪贴画

- 选择插入标签»剪贴画
- 单击剪贴画将打开搜索框如下屏幕截图。从可用剪贴画的列表
- 点击剪贴画以将其插入工作表



插入艺术字

- 选择插入标签»艺术字
- 选择你喜欢的艺术字样式，然后单击它在它输入文本。

Excel交叉参考 - Excel教程

在MS Excel的图形对象

当你有信息分散在几个不同的电子表格，它似乎是一项艰巨的任务，把所有这些不同的数据集在一起成一个有意义的列表或表格。这是VLOOKUP函数进入它自己。

VLOOKUP

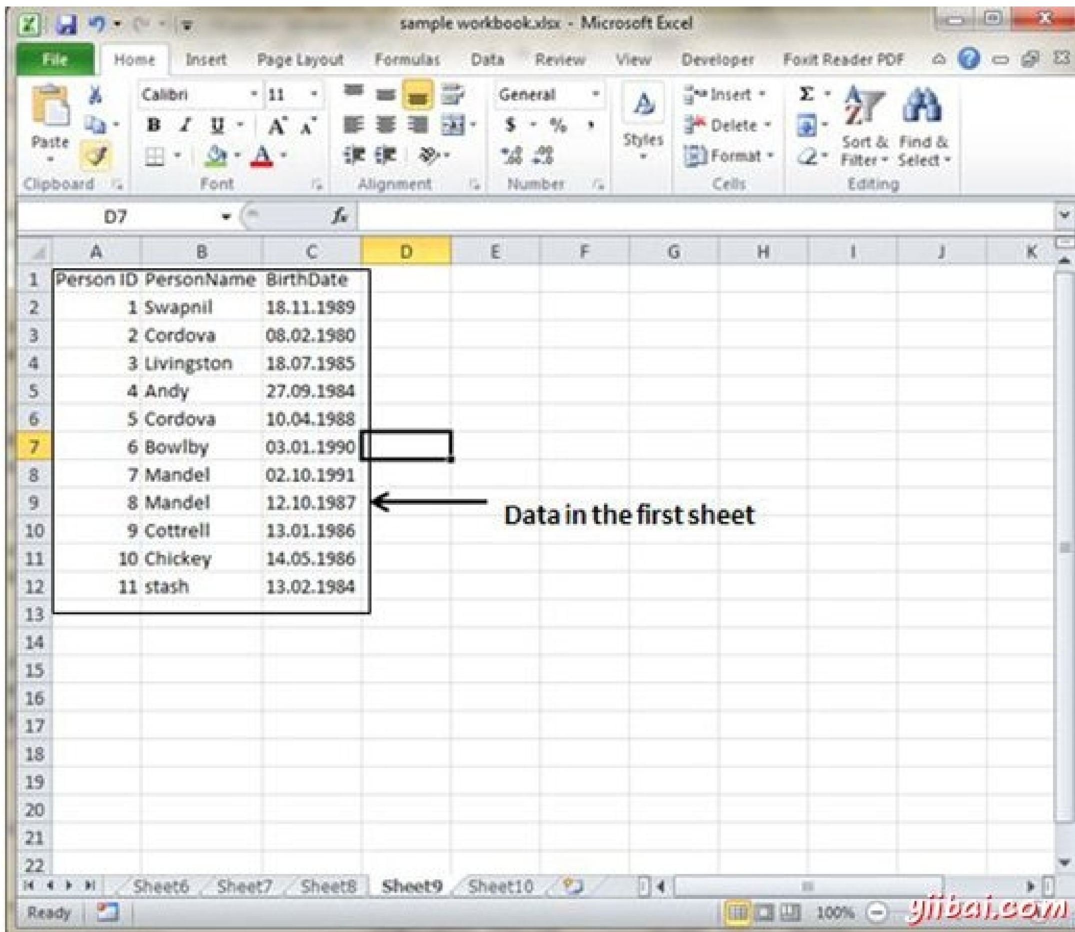
VLOOKUP搜索垂直向下的查找表的一个值。 VLOOKUP(lookup_value，table_array，Col_index_num，range_lookup)有4个参数如下。

- lookup_value : 它是用户输入。这是该函数用于搜索的值。
- The table_array : 它是表中单元格所在的区域。这不仅包括列被搜索，也会得到所需要的值的数据列。
- Col_index_num : 这是数据包含想要的答案列。
- Range_lookup : 这是一个TRUE或FALSE值。当设置为TRUE，查找函数给出最接近lookup_value，而无需在lookup_value。当设置为false，精确匹配，必须找到对lookup_value或函数将返回＃N / A。注意， 这需要包含lookup_value列按升序进行格式化。

VLOOKUP 例子

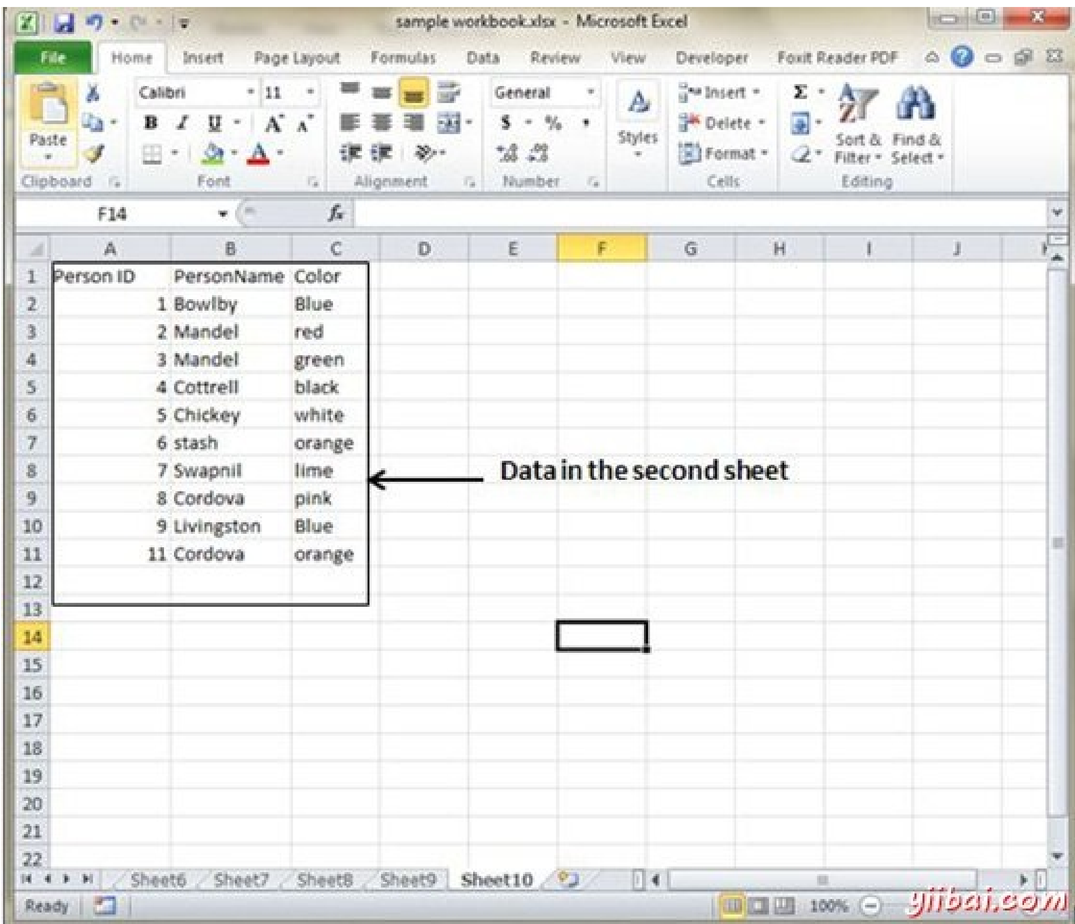
让我们来看看交叉参考两幅电子表格的一个很简单的例子。每个电子表格包含了同一组人的信息。第一个电子表格有他们的出生日期，而第二显示自己喜欢的颜色。我们如何建立一个列表，显示这个人的姓名，出生日期和自己喜欢的颜色。VLOOOKUP将有助于在这种情况下。首先让我们来看看在这两个表的数据。

这是在第一个电子表的数据



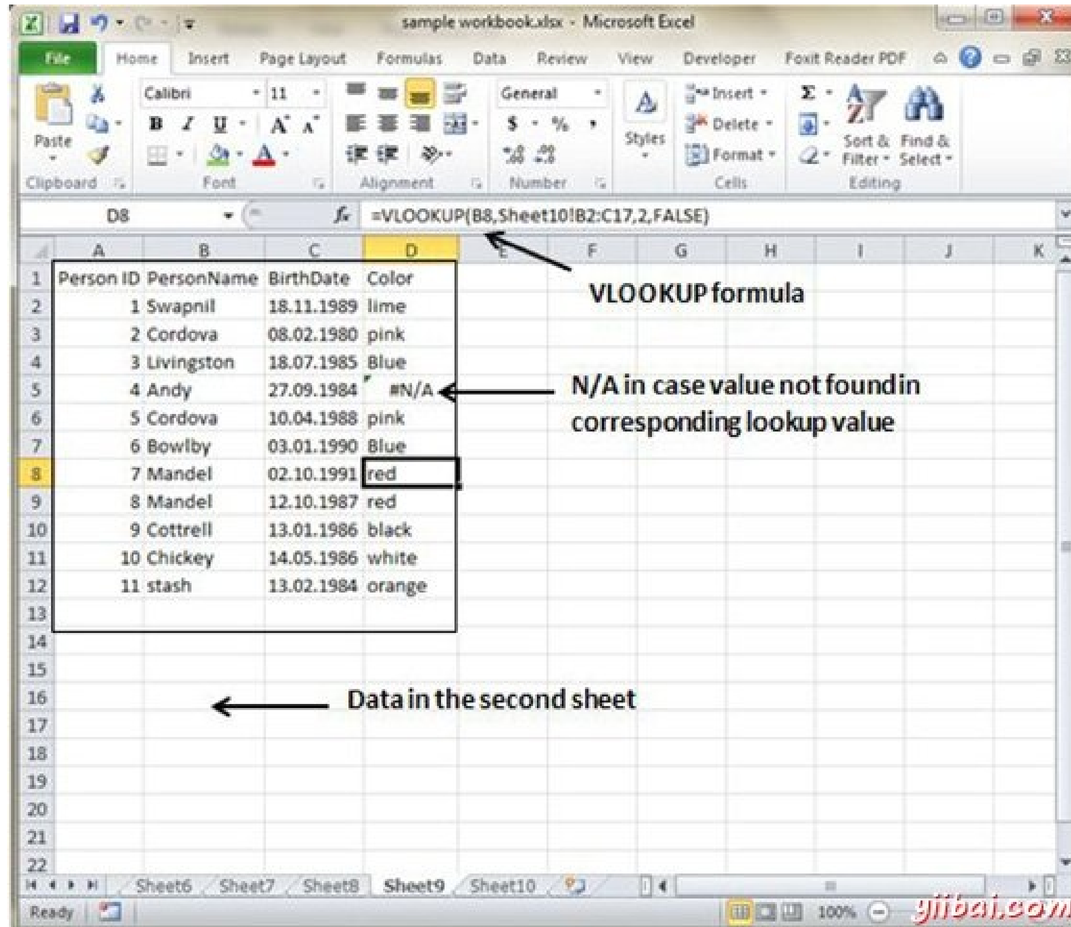
Person ID	PersonName	BirthDate
1	Swapnil	18.11.1989
2	Cordova	08.02.1980
3	Livingston	18.07.1985
4	Andy	27.09.1984
5	Cordova	10.04.1988
6	Bowlby	03.01.1990
7	Mandel	02.10.1991
8	Mandel	12.10.1987
9	Cottrell	13.01.1986
10	Chickey	14.05.1986
11	stash	13.02.1984

这是在第二个电子表的数据



Person ID	PersonName	Color
1	Bowlby	Blue
2	Mandel	red
3	Mandel	green
4	Cottrell	black
5	Chickey	white
6	stash	orange
7	Swapnil	lime
8	Cordova	pink
9	Livingston	Blue
11	Cordova	orange

现在，从另一片找到相应喜爱的颜色，我们需要VLOOKUP数据。第一个参数是VLOOKUP查找值（在这种情况下，它是人的名字）。第二个参数是在表阵列中，表中第二板从B2到C11。 第三个参数是VLOOKUP列索引NUM这就是回答，我们所期待的。在这种情况下，它是2的颜色的列数为2。 第四个参数是真返回部分匹配或假返回精确匹配。应用VLOOKUP公式之后，将计算出的颜色和结果显示如下。



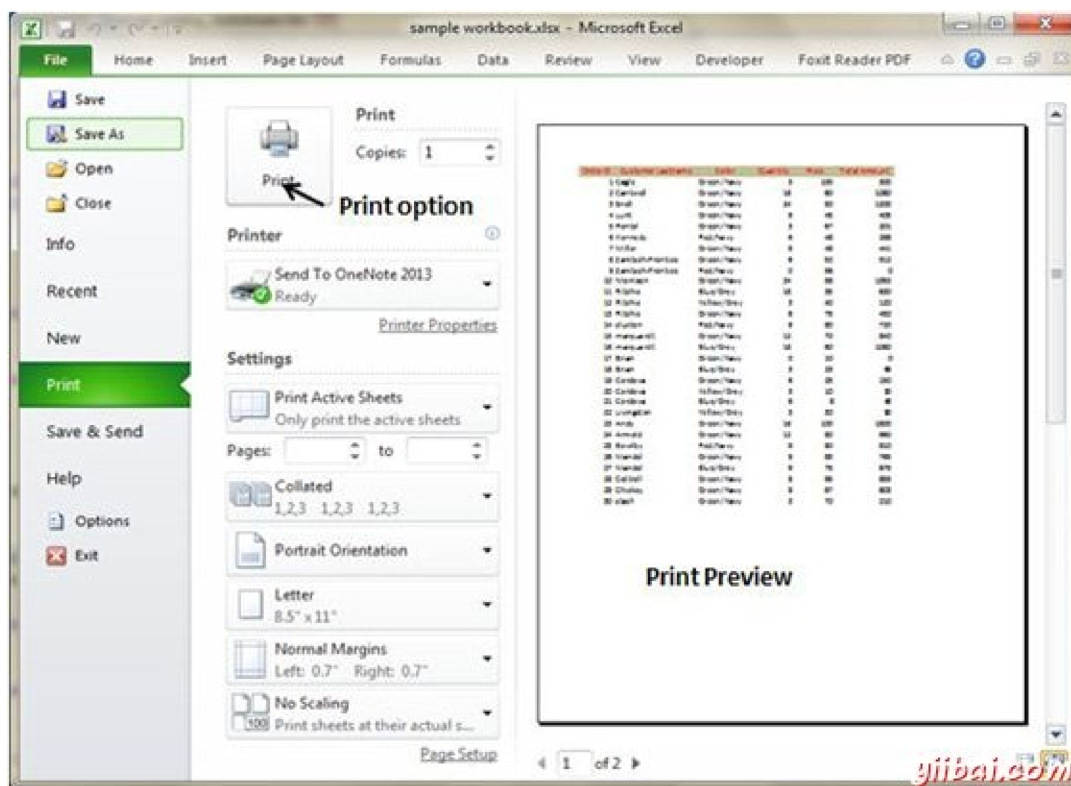
正如你可以在上面的屏幕截图的VLOOKUP结果已搜索颜色第二张表中看到。在未找到匹配它返回＃N / A的情况下。在这种情况下，Andy的数据不存在于所述第二电子表格，以便它返回＃N / A。

Excel打印工作表 - Excel教程

快速打印

如果你要打印工作表的副本没有布局的调整，使用快速打印选项。我们可以使用此选项有两种方式。

- 选择文件»打印(其中显示了打印窗格)，然后点击打印按钮。
- 按下Ctrl + P键，然后点击打印按钮(或按Enter键)。



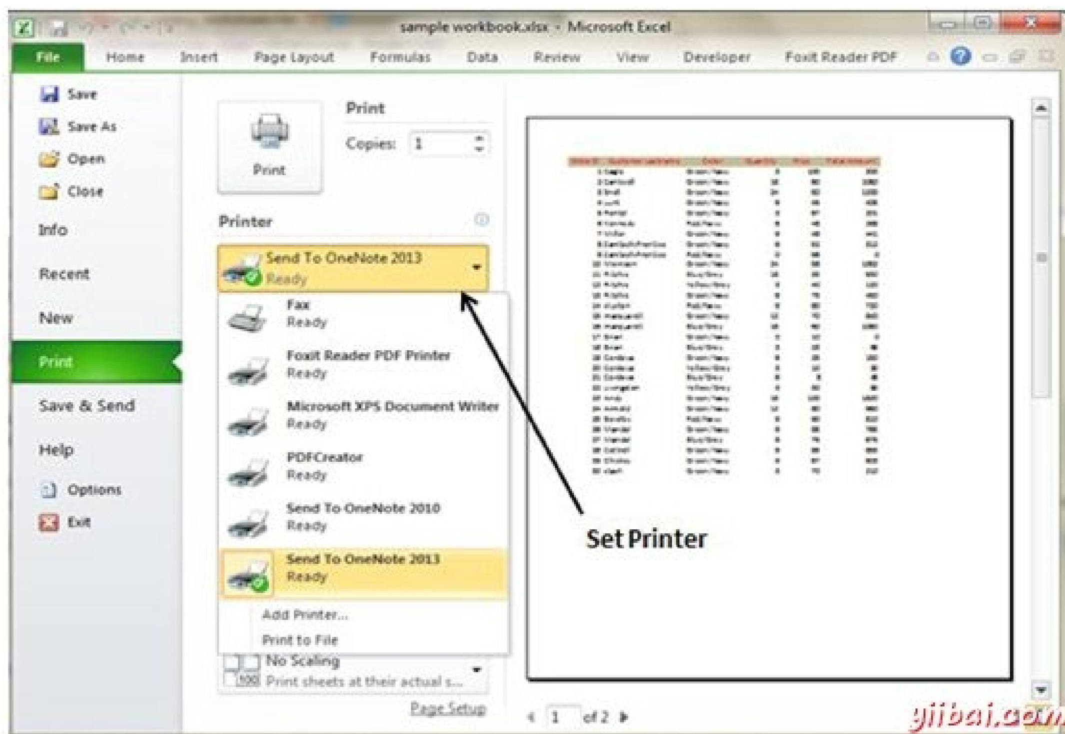
常见的调整页面设置

您可以调整在不同的方式如下页面设置对话框提供的打印设置。页面设置选项包括页面方向，页面尺寸，页边距等。

- 在Backstage视图打印屏幕，显示当你选择 文件»打印
- 功能区的页面布局选项卡

选择您的打印机

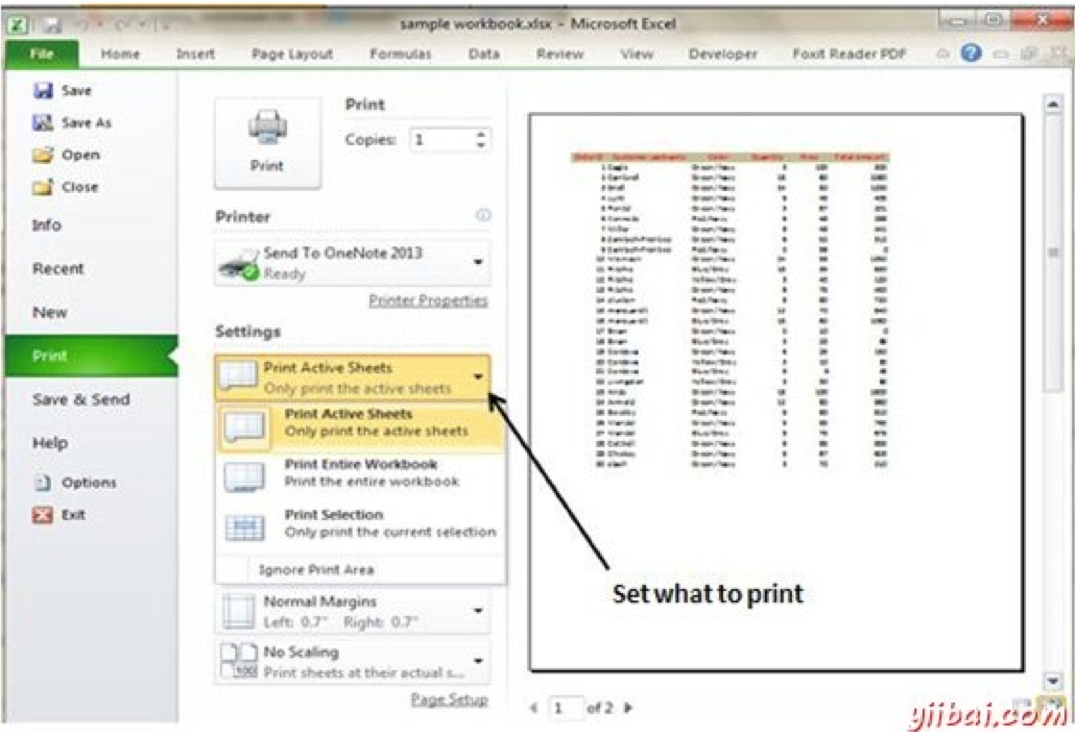
要切换到不同的打印机，选择文件»打印并使用下拉控制在打印机部分选择不同的安装的打印机。



指定要打印的内容

有时你可能需要打印工作表，而不是整个活动区域中的一部分。选择文件»打印并使用控件设置部分指定要打印的内容。

- 活动表：打印所选的活动工作表或表
- 整个工作簿：打印整个工作簿，包括图表工作表
- 选择：只打印选择文件»打印前选择的范围

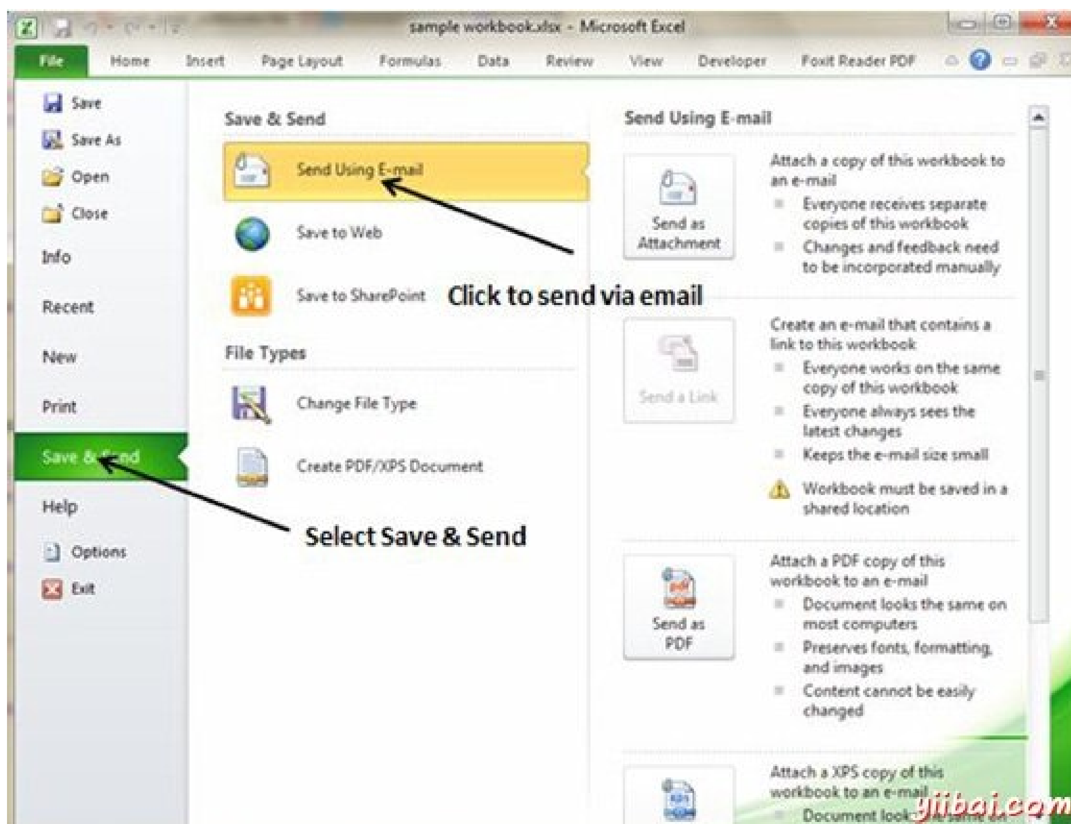

sample workbook.xlsx - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Foxit Reader PDF
Save
Save As
Open
Close
Info
Recent
New
Print
Save & Send
Help
Options
Exit
Print
Copies: 1
Printer
Send To OneNote 2013
Ready
Printer Properties
Settings
Print Active Sheets
Only print the active sheets
Print Entire Workbook
Print the entire workbook
Print Selection
Only print the current selection
Ignore Print Area
Normal Margins
Left: 0.7" Right: 0.7"
No Scaling
Print sheets at their actual s...
Page Setup
Set what to print
1 of 2
yiibai.com

Excel电子邮件簿 - Excel教程

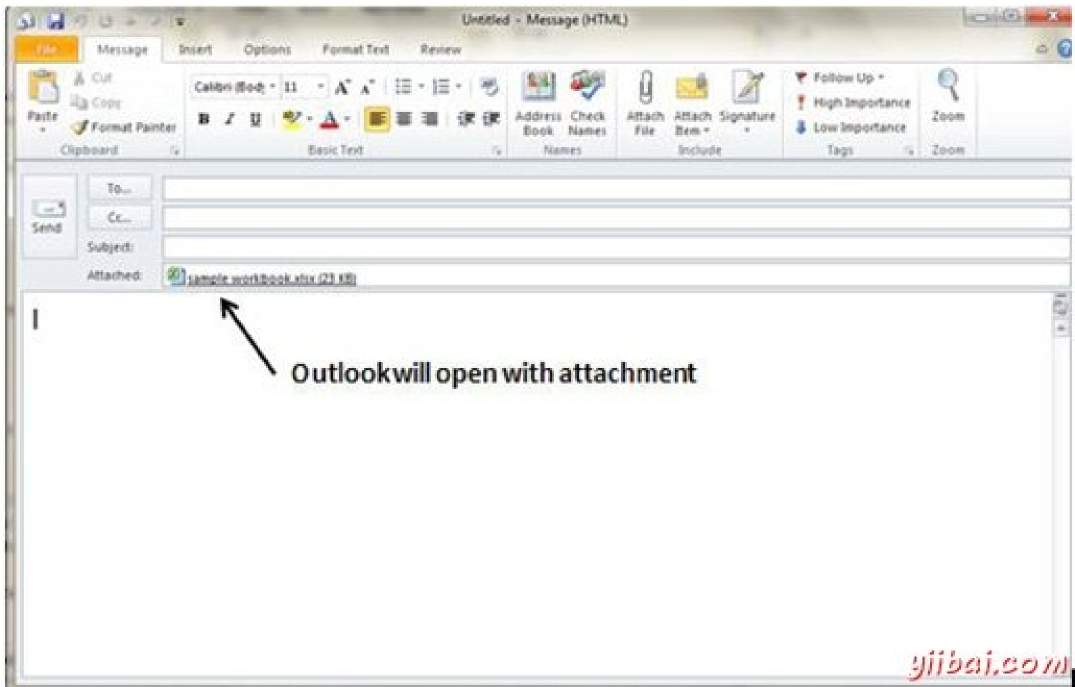
电子邮件簿

MS Excel中，您可以非常容易地通过电子邮件发送工作簿。要通过电子邮件发送工作簿给任何人请按以下步骤操作。

- 选择文件»保存和发送。它基本上是第一个，然后保存文档后发送电子邮件。



- 点击使用E-mail发送。如果您的电子邮件系统有配置。MS Outlook将与文件作为附件的电子邮件的新窗口中打开。您可以将这个工作簿发送邮件给任何人，在有效的电子邮件地址。


Untitled - Message (HTML)
File
Message
Insert
Options
Format Text
Review
Paste
Cut
Copy
Format Painter
Clipboard
Basic Text
Address Book
Check Names
Names
Attach File
Attach Item
Signature
Include
Follow Up
High Importance
Low Importance
Tags
Zoom
Send
To...
Cc...
Subject:
Attached:
Outlookwill open with attachment
yiibai.com

Excel翻译工作表 - Excel教程

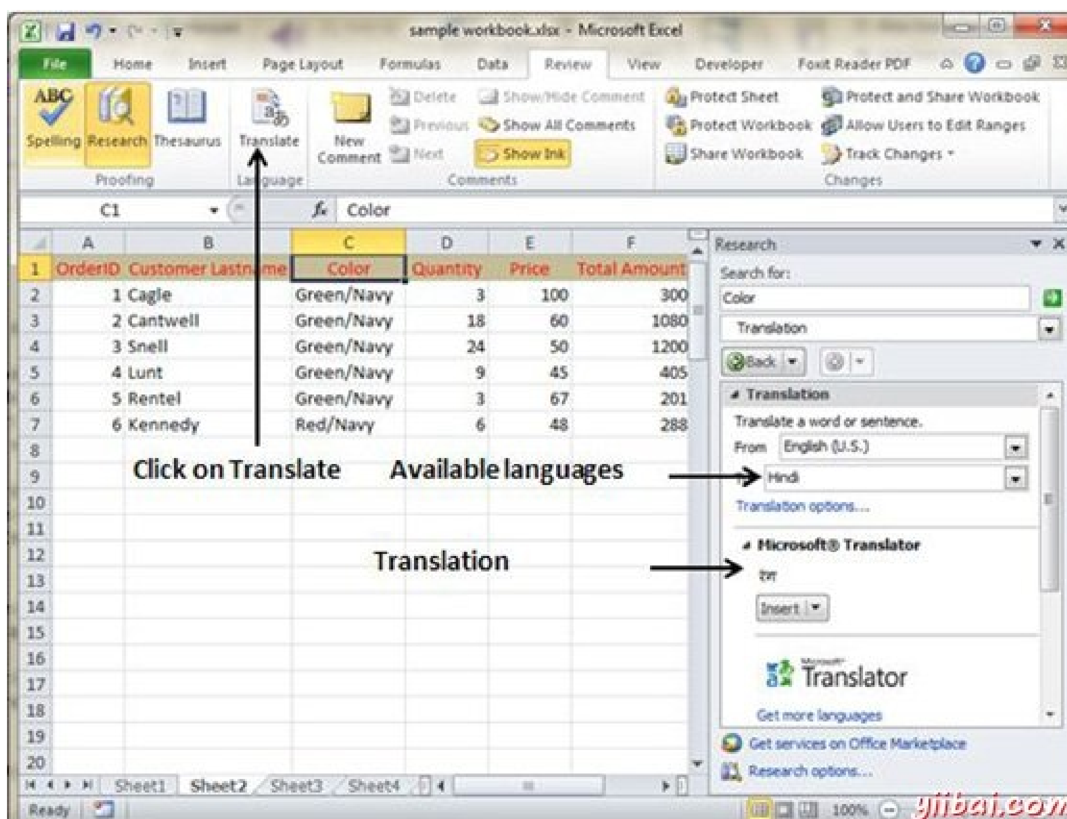
Excel翻译工作表

你可以翻译文本写在不同的语言，比如词组或段落，个别字(通过使用迷你翻译)，或使用MS Excel2010转换整个文件。

翻译可用在MS Excel 2010色带选项中，您可以迅速转化成单元不同的语言与此选项的审查选项卡可用。

翻译一步一步完成

- 选择您想要翻译成不同的语言的内容。
- 选择审查标签»翻译。
- 这将打开您可以从中选择需要翻译的语言窗格。
- 需要互联网连接进行转换。这转化使用微软翻译。
- 点击插入到应用平移更改。



Excel工作簿安全 - Excel教程

工作簿安全

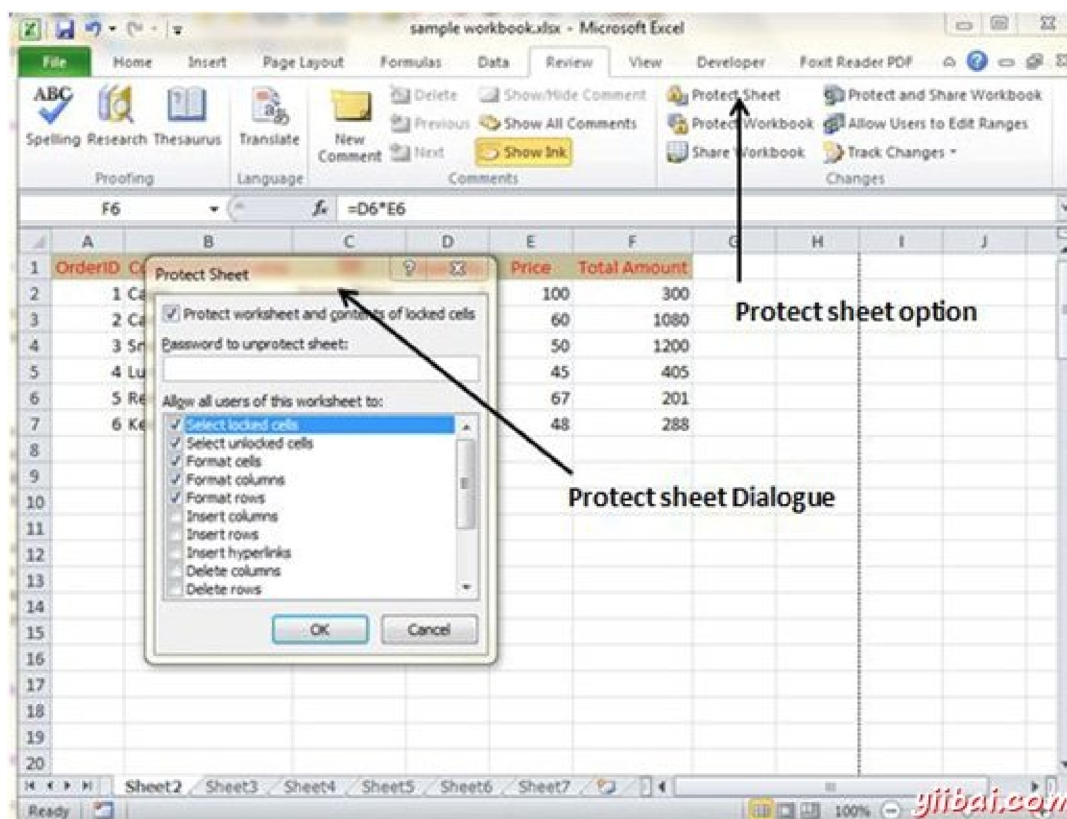
我们可以通过在色带的审查选项卡中提供保护的概念应用安全工作簿。MS Excel保护相关的功能分为三类。

- 工作表保护：被修改保护工作表，或限制修改某些用户。
- 工作簿保护：从具有插入或删除表，并且还需要使用密码来打开工作簿保护工作簿

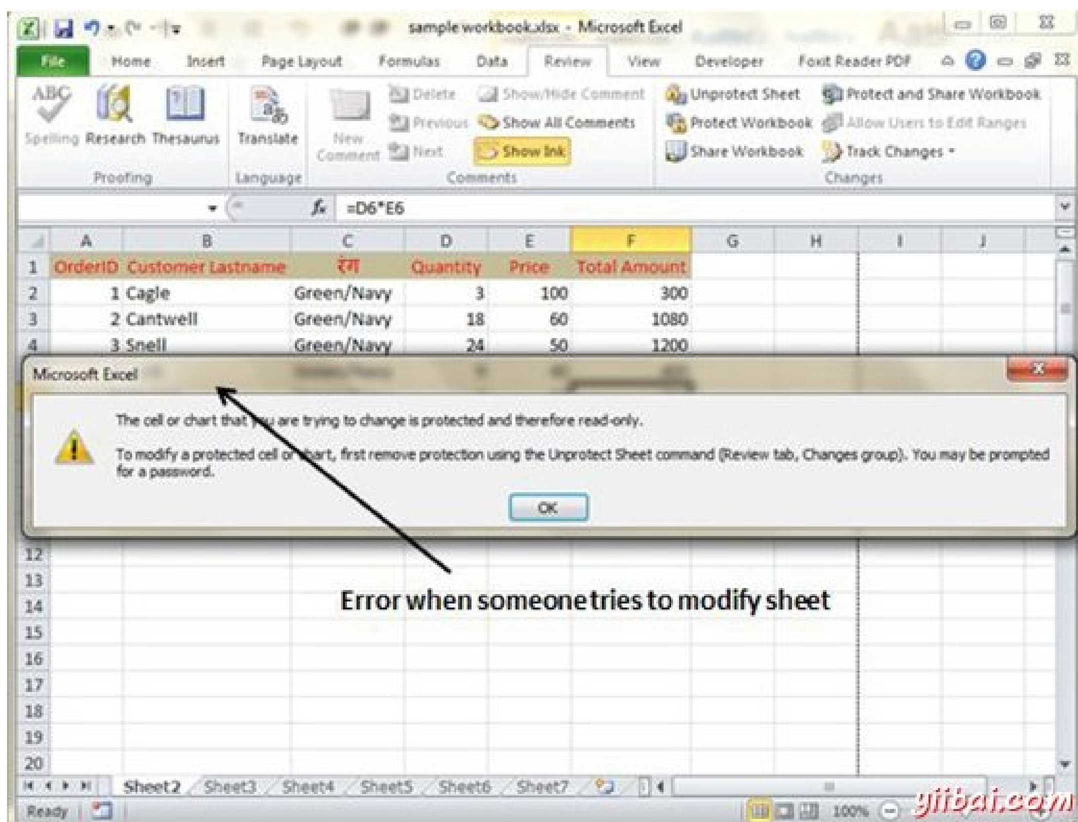
保护工作表

可能想要保护工作表出于各种原因。其中一个原因是为了防止自己或他人意外删除公式或其他关键数据。 一个常见的情况是，以保护一个工作表，使得数据可以被改变，但公式不能被改变。

为了保护工作表，选择评论»更改组»保护工作表。Excel显示保护工作表对话框。需要注意的是，提供密码是可选的。如果你输入密码，密码将用于取消工作表。可以选择的各种选项，其工作表受到保护。假设我们选中单元格格式选项，则Excel将不允许设置单元格格式。



当有人试图单元格格式化，会得到如下错误。



要解除保护工作表，选择审查»更改组»撤消工作表保护。如果工作表受密码保护，系统会提示您输入密码。

保护工作簿

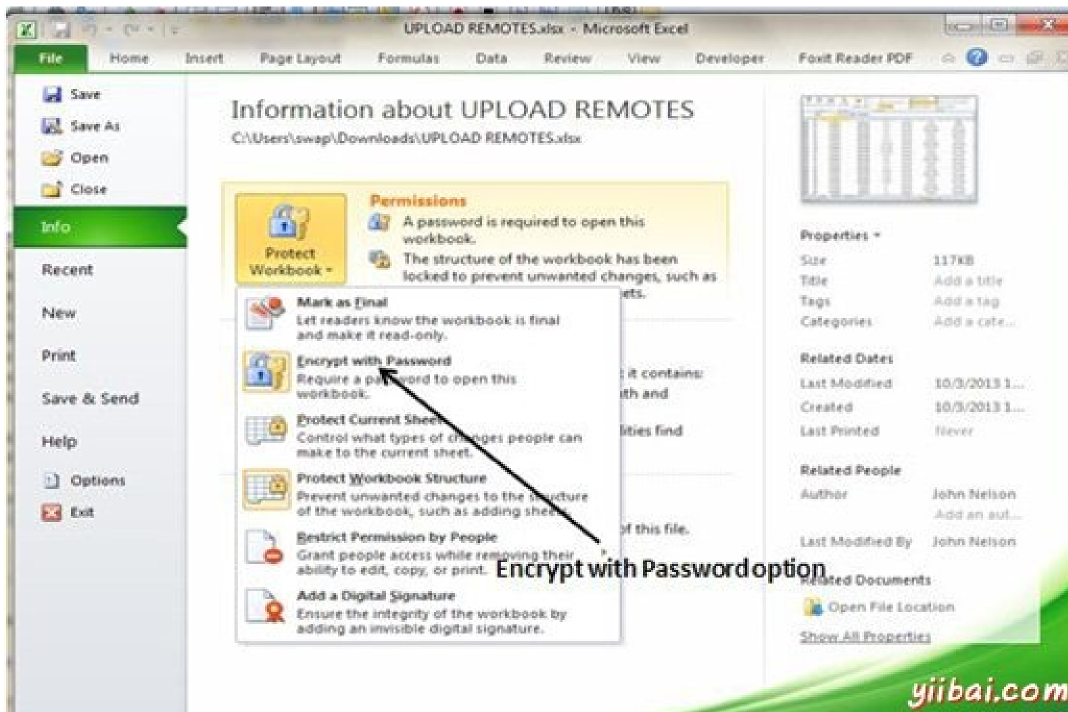
Excel提供了三种方法来保护工作簿。

- 需要密码才能打开工作簿。
- 防止用户从添加表，删除表，隐藏表和取消隐藏工作表。
- 阻止用户从改变窗口的大小或位置。

需要密码来打开工作簿

Excel中可以让你使用密码保存工作簿。这样做之后，任何人试图打开工作簿时必须输入密码。要将密码添加到工作簿，请按照下列步骤。

- 选择[文件]»信息»保护工作簿»加密，具有密码。 Excel显示加密文档对话框。
- 键入密码，然后单击确定。
- 再次键入密码，然后单击确定。
- 保存工作簿。



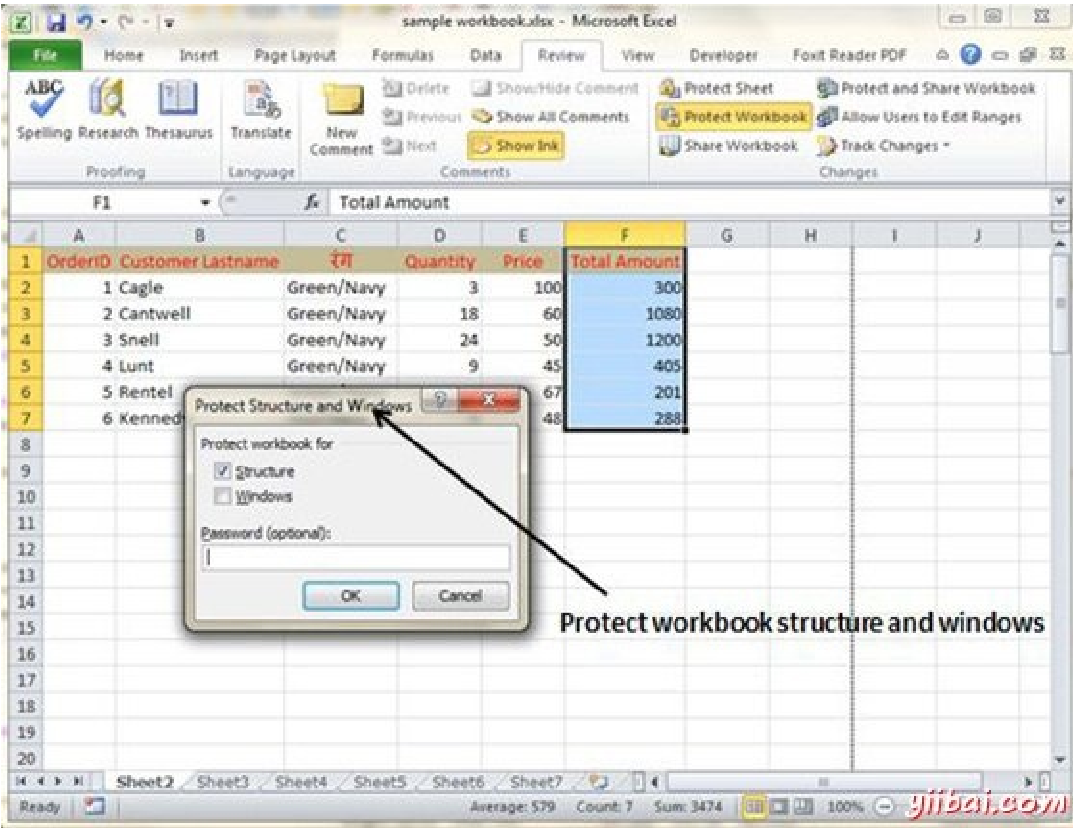
要从工作簿中删除密码，重复相同的过程。在步骤2中，不过删除现有的密码符号。

保护工作簿的结构和Windows

为了防止他人（或自己）在工作簿中执行某些操作，可以保护工作簿的结构和窗口。当一个工作簿的结构和窗口被保护，则用户可能不添加工作表，删除工作表，隐藏工作表，取消隐藏工作表等，并可能不允许分别改变工作簿的视窗大小或位置。

要保护工作表的结构和Windows，按照以下步骤

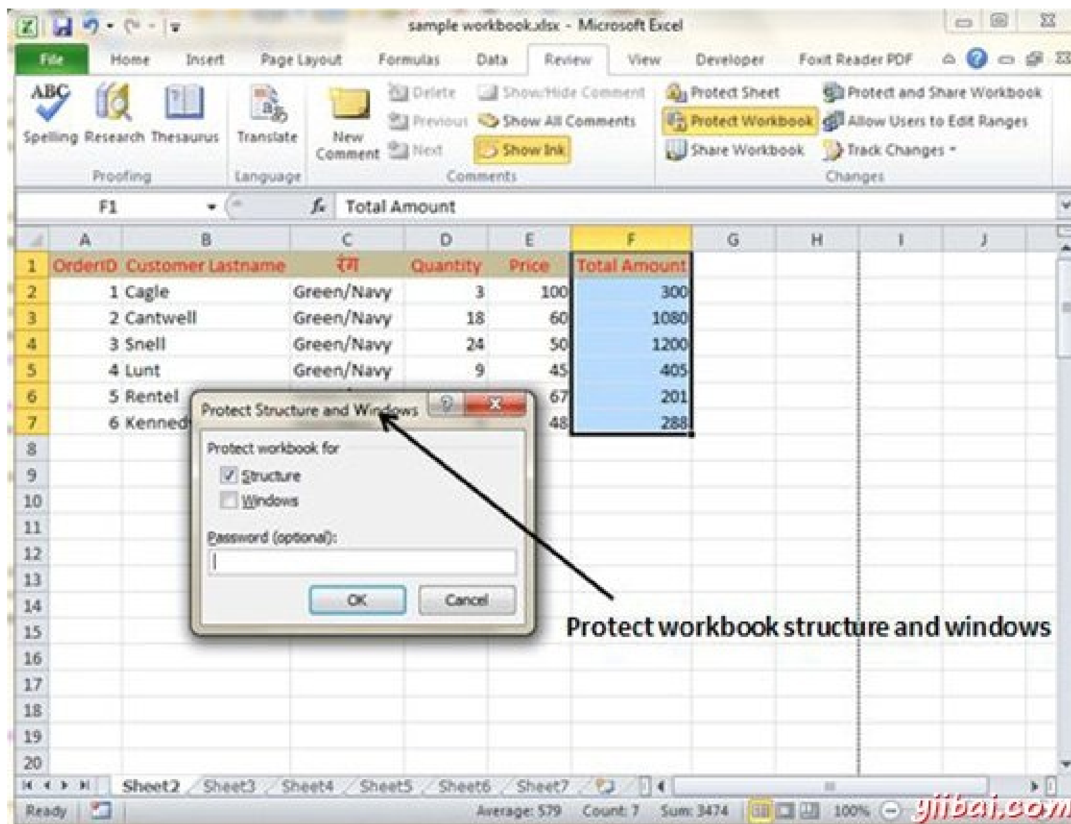
- 选择审查»更改组»保护工作簿，以显示保护工作簿对话框。
- 在保护工作簿对话框中，选中结构复选框和Windows复选框。
- （可选）输入密码。
- 点击确定。


sample workbook.xlsx - Microsoft Excel
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Foxit Reader PDF
Spelling
Research
Thesaurus
Translate
New Comment
Delete
Previous
Next
Show/Hide Comment
Show All Comments
Show Ink
Protect Sheet
Protect Workbook
Share Workbook
Protect and Share Workbook
Allow Users to Edit Ranges
Track Changes
Proofing
Language
Comments
Changes
F1
Total Amount
OrderID
Customer Lastname
Quantity
Price
Total Amount
1 Cagle
2 Cantwell
3 Snell
4 Lunt
5 Rentel
Green/Navy
Protect Structure and Windows
Protect workbook for
Structure
Windows
Password (optional):
OK
Cancel
Protect workbook structure and windows
Sheet2
Sheet3
Sheet4
Sheet5
Sheet6
Sheet7
Ready
Average: 579
Count: 7
Sum: 3474
100%
yiibai.com

Excel数据表 - Excel教程

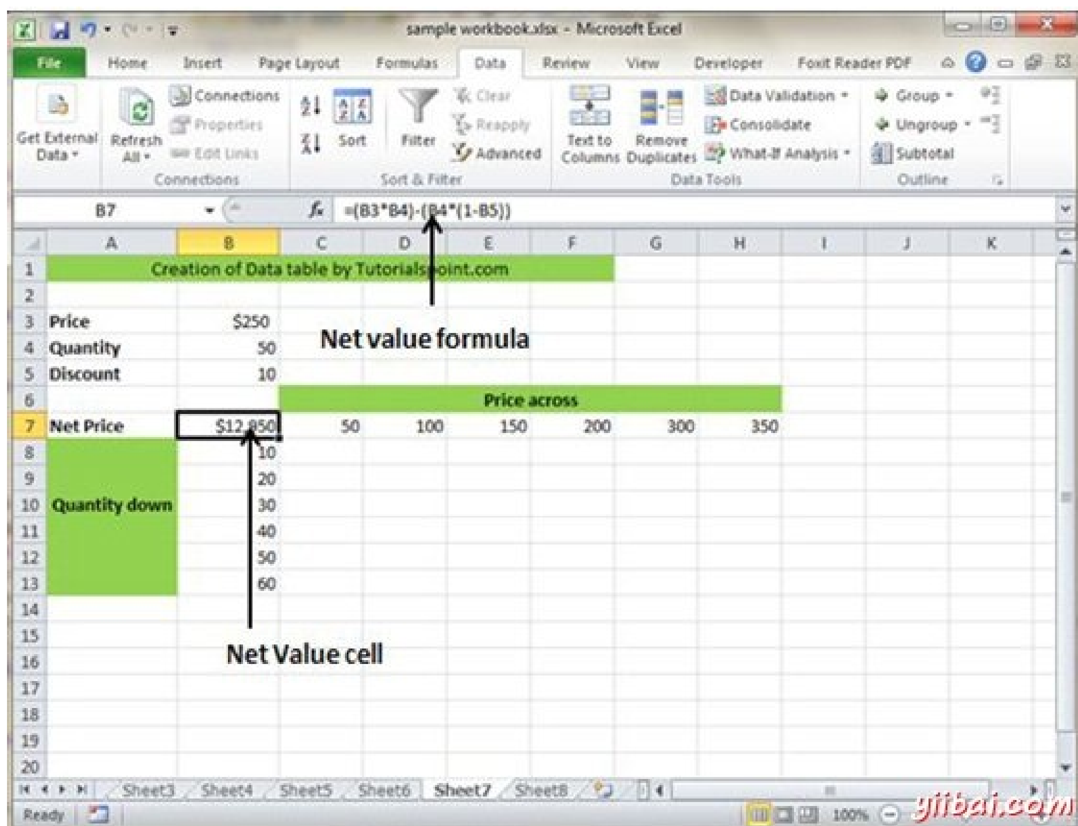
数据表

在Excel中，一个数据表是一种通过改变公式输入单元格，而看到不同的结果。 数据表中提供在数据选项卡»假设分析下拉»数据表 在MS Excel

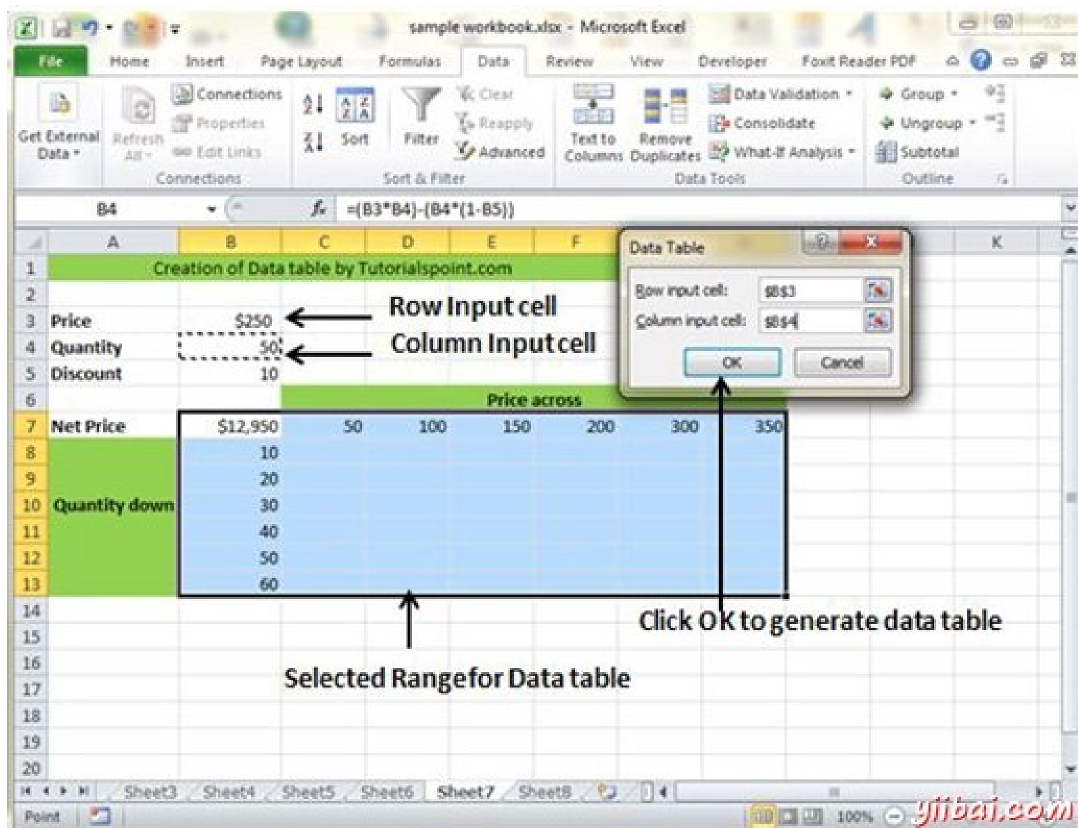


数据表实例

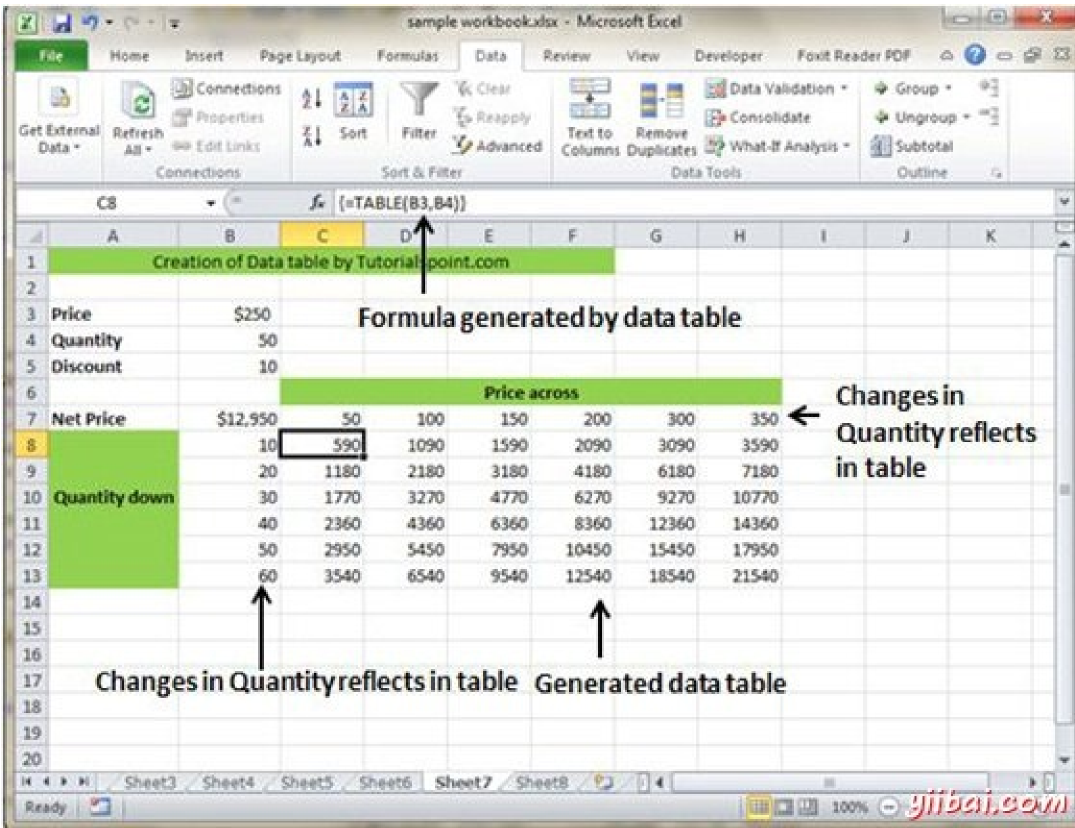
现在，让我们来看看数据表的概念，这里用一个例子来说明。假设有价格及数量的多个值。也由折扣计算净价，作为第三个变量。 你可以保持净价值在数据表的帮助下组织表格式。价格横向运行，以正确的数量，同时运行垂直向下。 我们正在使用的公式来计算净价格为价格乘以数量减去总折扣(数量 x 折扣)



现在制作数据表中的选择数据表的范围。 选择数据选项卡»假设分析下拉»数据表。它会出现对话要求输入行和输入列。得到的输入行作为价格单元格(在这种情况下单元为B3)和作为数量单元格的输入列(在这种情况下，单元格为B4)。请参见下面的屏幕截图。



如示于下面的屏幕截图点击确定将产生的数据表。它会生成表公式。您可以更改横向的价格或数量垂直的净价看到变化。



Excel数据透视表 - Excel教程

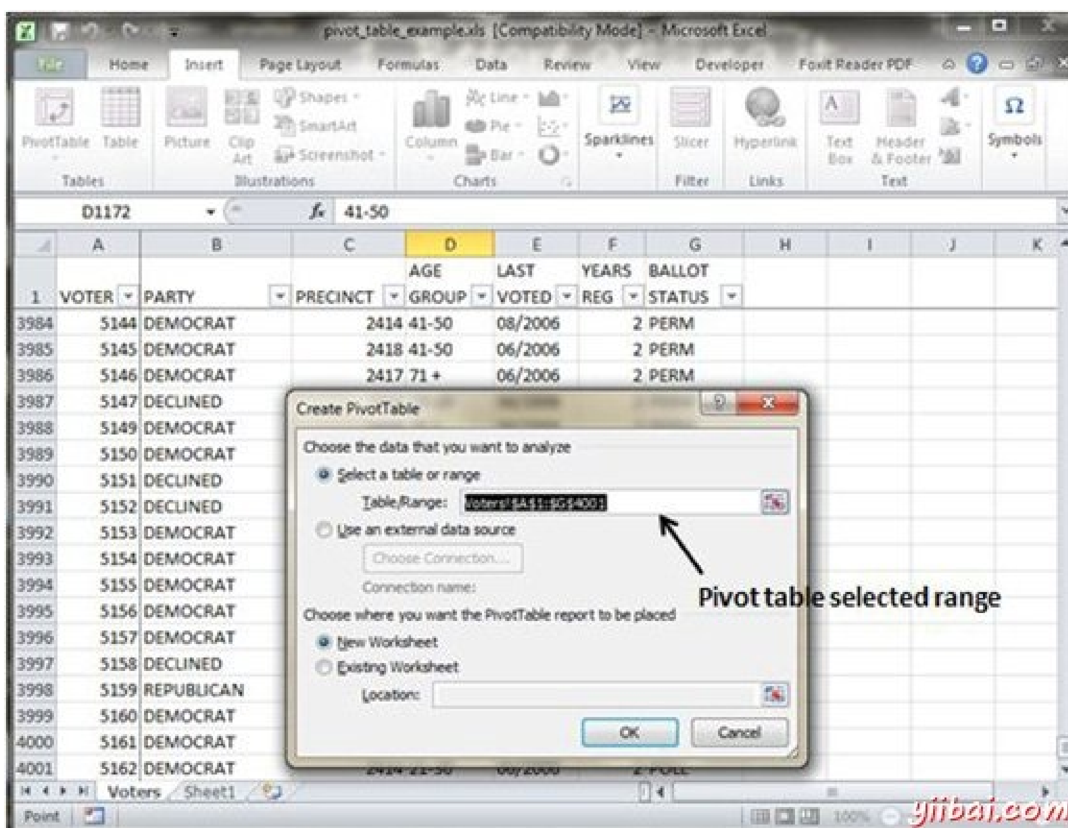
数据透视表

数据透视表实质上是从数据库中生成的动态汇总报告。数据库可以存在于一个工作表(表中的形式)，或在外部数据文件。数据透视表可以帮助改变一排排和数字列到一个有意义的数据呈现。 数据透视表是非常强大的工具，为数据的汇总分析。

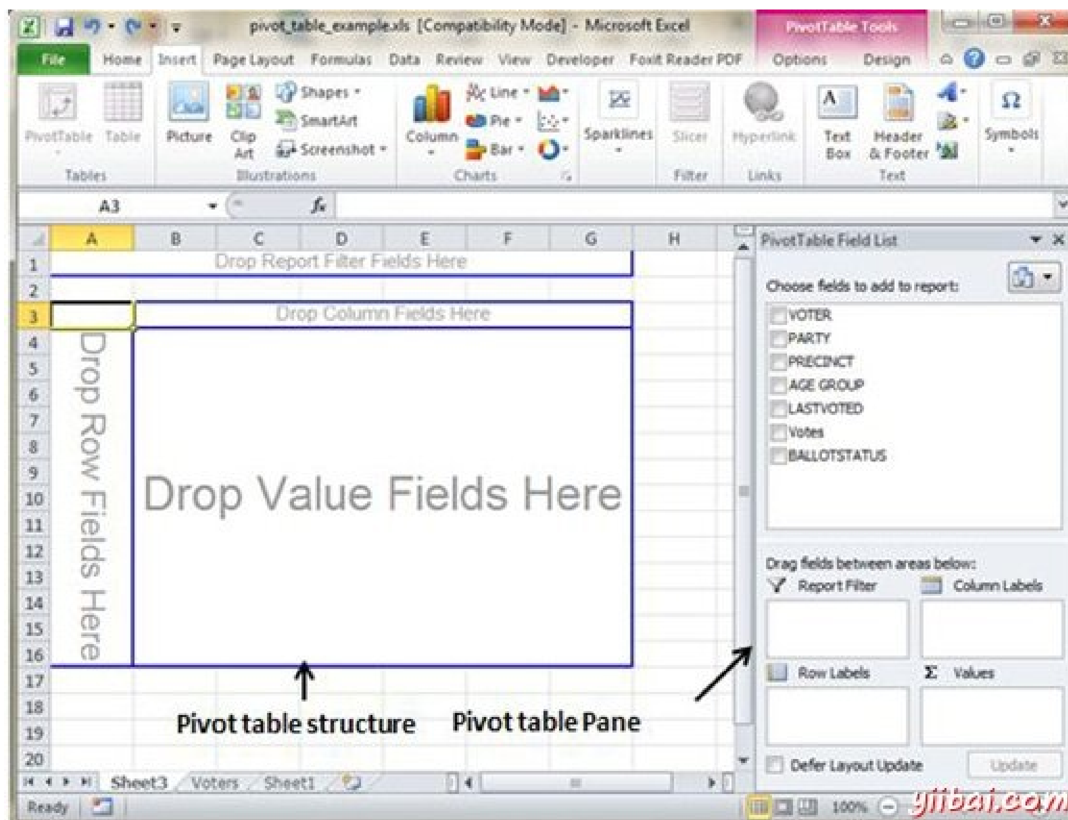
数据透视表是在插入选项卡»数据透视表下拉»透视表

数据透视表范例

现在，让我们来看看数据透视表的例子。假设你有选民庞大的数据，你希望看到每方选民信息的汇总数据，那么可以使用数据透视表。选择插入标签»数据透视表插入数据透视表。MS Excel选择表的数据。可以为现有表或新表选择透视表的位置。

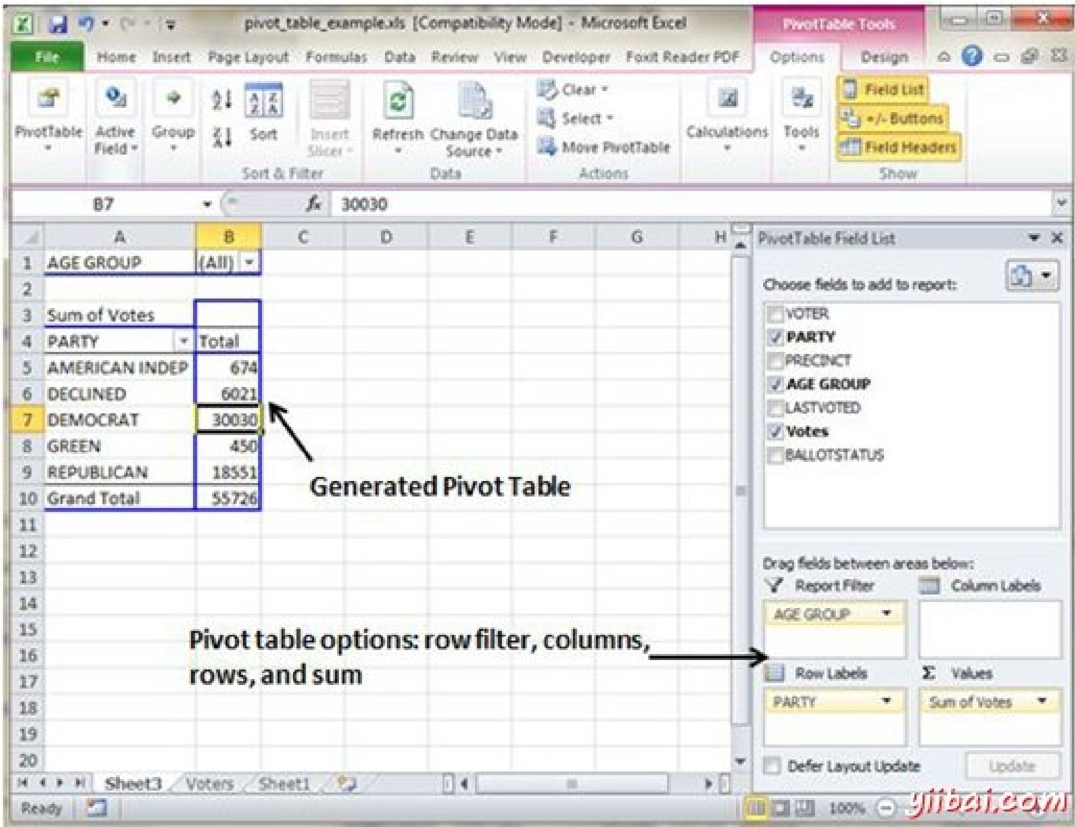


这将生成数据在透视表窗格中，如下图所示。有如下透视表窗格提供的各种选项。您可以选择生成的数据透视表的字段。



- 列标签：有一列方向在数据透视表的字段。字段的每一项占据一列。
- 报表过滤器：您可以设置报表的过滤器，然后每年得到的数据按年份过滤。
- 行标签：具有行方向在数据透视表中的字段。字段每一项占据一行。
- 值范围：单元格透视表包含汇总数据。Excel提供了几种方法来汇总数据（求和，平均值，计数等等）。

以数据透视表给输入字段后，它将产生一个与数据如下数据透视表。



pivot_table_example.xls [Compatibility Mode] - Microsoft Excel
PivotTable Tools
AGE GROUP (All)
Sum of Votes
PARTY Total
AMERICAN INDEP 674
DECLINED 6021
DEMOCRAT 30030
GREEN 450
REPUBLICAN 18551
Grand Total 55726
Generated Pivot Table
Pivot table options: row filter, columns, rows, and sum
PivotTable Field List
Choose fields to add to report:
VOTER
PARTY
PRECINCT
AGE GROUP
LASTVOTED
Votes
BALLOTSTATUS
Drag fields between areas below:
Report Filter
Column Labels
AGE GROUP
Row Labels
Values
PARTY
Sum of Votes
Defer Layout Update
Update

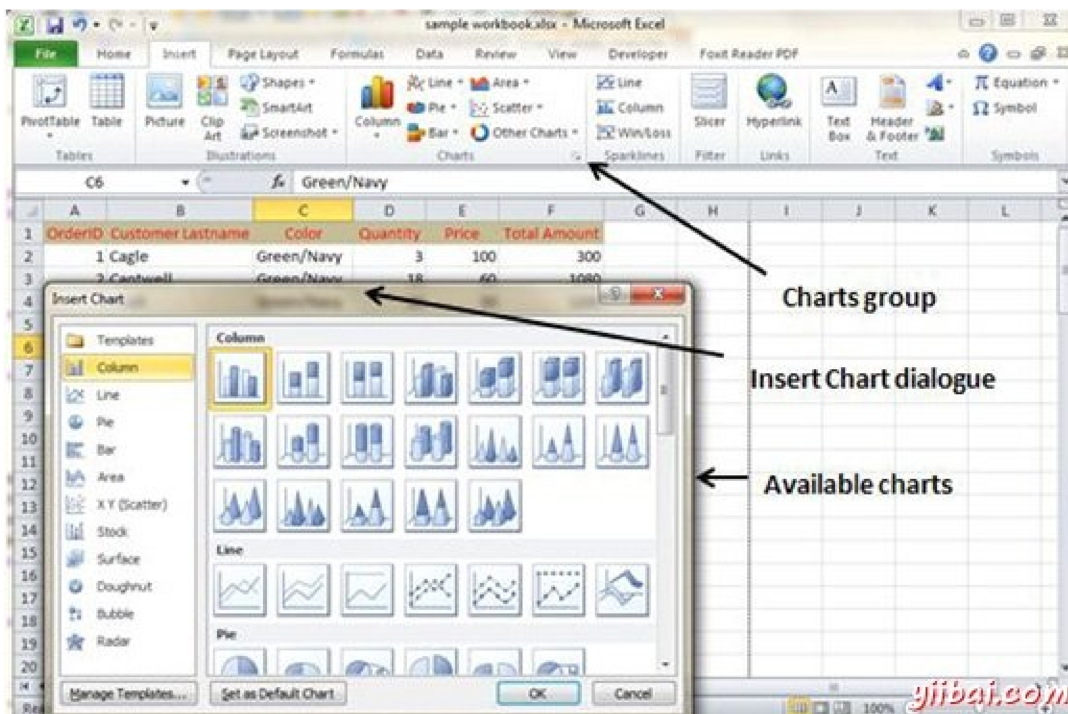
Excel图表 - Excel教程

图表

图表是数值的可视化表示。图表(也称为图形)已成为电子表格的一个组成部分。早期的电子表格产品生成的图表是相当粗糙，但这么多年来有所显著的改善。 Excel为您提供的工具来创建各种高度可定制的图表。数据显示在一个周密的图表可以让数字更容易理解。因为图表呈现图像，图表是用于概括的一系列数字和它们的相互关系特别有用的。

图表类型

在MS Excel有各种可用的图表类型，如下面的屏幕快照。



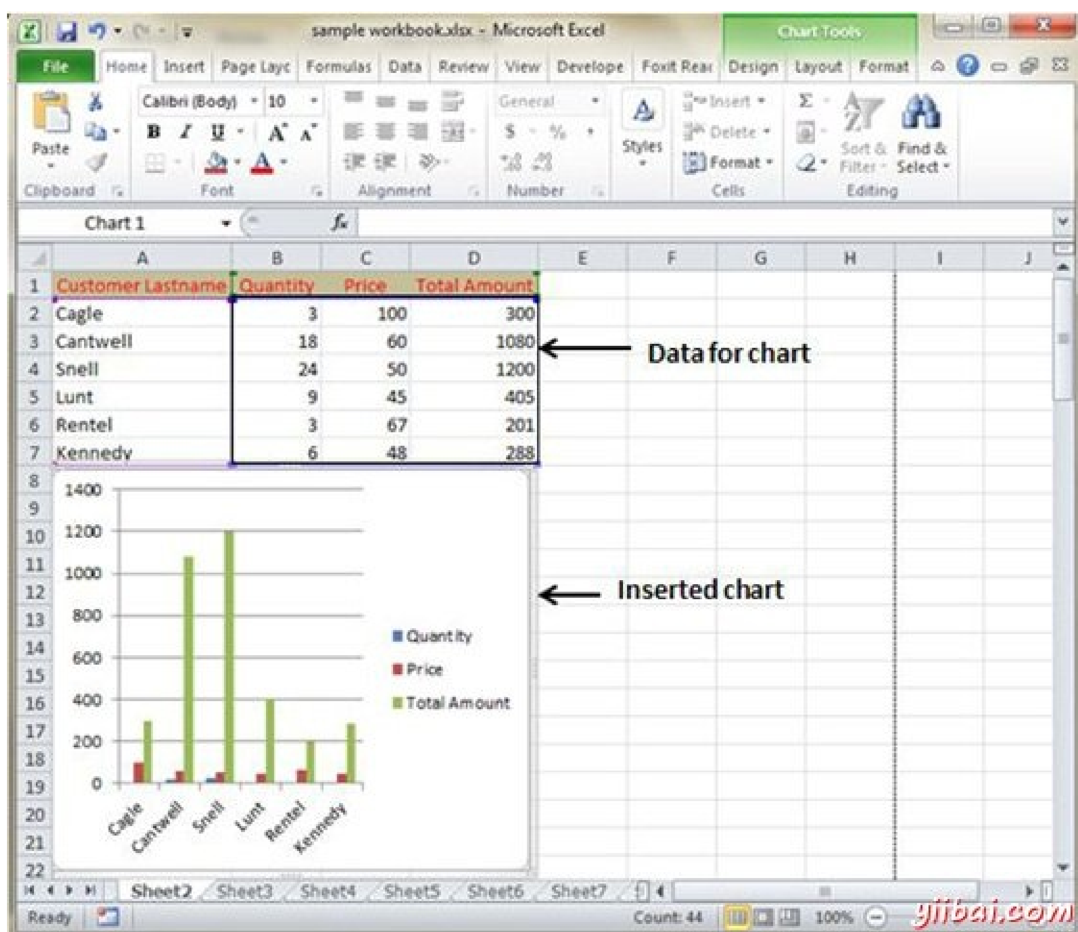
- 柱：柱形图显示了一段时间的数据变化或图示项目之间的比较。
- 块：图表块项目之间的比较。
- 饼图：饼图显示项目构成一个数据系列，成比例的项目总和的大小。它总是只显示一个数据系列，当想强调的一个数据元素显著是非常有用的。
- 线图：线图显示趋势在相同时间的数据。
- 区块：区块图强调变化的幅度随着时间的推移。

- **X Y 离散：**一个XY散点图显示中的几个数据系列的数值，或者地块两组数字为一体的系列XY坐标的关系。
- 股票：这种图表类型是最经常使用的股价数据，但也可用于科学数据（例如，以指示温度变化）。
- 曲面：当想找到两组数据之间的最佳组合曲面图是有用的。 如在地形图，颜色和图案表明，在相同的值范围的区域。
- 圆环图：像一个饼图，圆环图显示部分到整体的关系; 不过，它可以包含一个以上的数据系列。
- 气泡：被布置成列的工作表上，以便使x值列在第一列和相应的y值和气泡大小值列于相邻的列中的数据，可以绘制在气泡图。
- 雷达：雷达图比较多个数据系列的合计值。

创建图表

创建图表由下面的步骤数据。

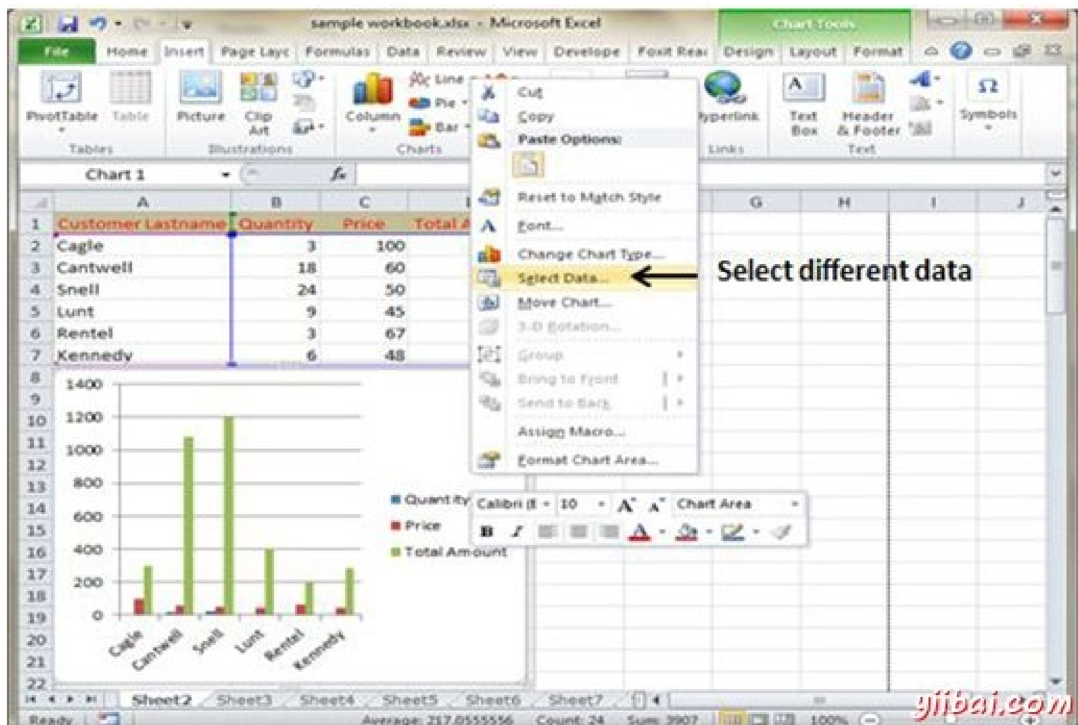
- 选择您想要创建图表的数据。
- 选择插入标签»选择图表或单击图表组看到各种图表类型。
- 选择您所选择的图表，然后单击确定生成图表。



编辑图表

在已经创造了它之后，在任何时候，可以编辑图表。

- 可以选择不同的数据，图表输入，右键单击图表»选择数据。 选择新的数据将产生如在下面的屏幕截图图表按新的数据。



- 可以通过以图表的X轴给出不同的输入改变图表的X轴。
- 可以通过以图表的Y轴赋予不同的输入更改图表的Y轴。

Excel枢轴透视图表 - Excel教程

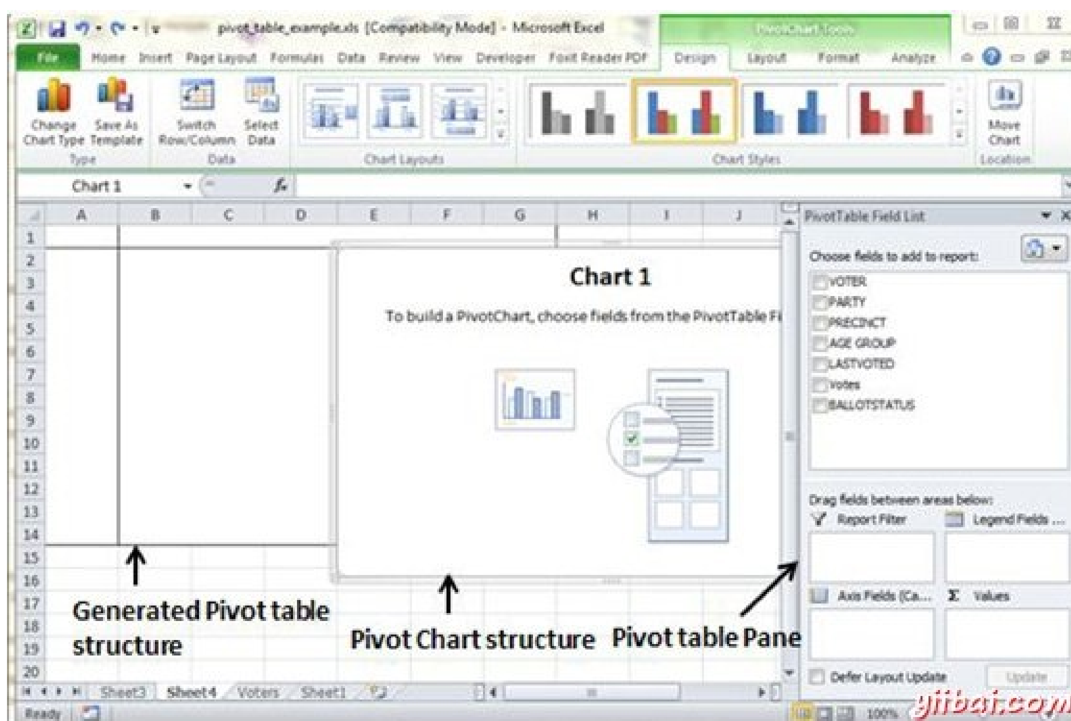
透视图表

枢轴表是透视表显示数据汇总的图形表示。枢轴表总是基于一个数据透视表。尽管Excel允许您创建一个数据透视表，并在同一时间一个数据透视图，但是不能创建一个没有数据透视表。 所有的Excel图表功能在一个数据透视图可用

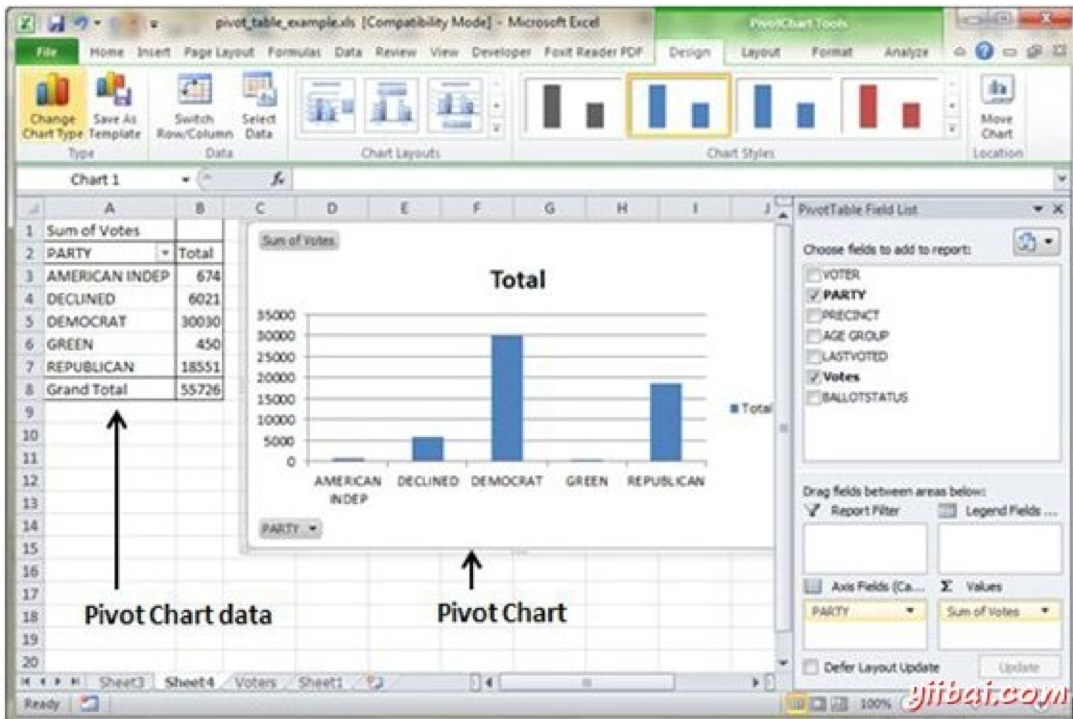
透视图是在插入 选项卡»透视表下拉»数据透视图可用

透视图表示例

现在，让我们来看看数据透视表的例子。假设你有选民庞大的数据，希望以图表形式显示，可以使用数据透视表为它的每形式选民信息的数据汇总视图。选择插入标签»透视图表插入数据透视表。



MS Excel选择表的数据。您可以从现有表或新表选择数据透视图的位置。数据透视图将取决于自动创建透视表在MS Excel中。可以在下面的屏幕截图中所生成的数据透视图


pivot_table_example.xls [Compatibility Mode] - Microsoft Excel
PivotChart Tools
File
Home
Insert
Page Layout
Formulas
Data
Review
View
Developer
Foxit Reader PDF
Design
Layout
Format
Analyze
Change Chart Type
Save As Template
Type
Switch Row/Column
Select Data
Data
Chart Layouts
Chart Styles
Move Chart
Location
Chart 1
Sum of Votes
PARTY
Total
AMERICAN INDEP
674
DECLINED
6021
DEMOCRAT
30030
GREEN
450
REPUBLICAN
18551
Grand Total
55726
Sum of Votes
Total
35000
30000
25000
20000
15000
10000
5000
0
AMERICAN INDEP
DECLINED
DEMOCRAT
GREEN
REPUBLICAN
Total
PARTY
PivotTable Field List
Choose fields to add to report:
VOTER
PARTY
PRECINCT
AGE GROUP
LASTVOTED
Votes
BALLOTSTATUS
Drag fields between areas below:
Report Filter
Legend Fields ...
Axis Fields (Ca...
Values
PARTY
Sum of Votes
Defer Layout Update
Update
Pivot Chart data
Pivot Chart
Sheet3
Sheet4
Voters
Sheet1
Ready
100%

Excel快捷键 - Excel教程

MS Excel的键盘快捷键

Excel中提供了许多键盘快捷键。如果你熟悉Windows操作系统就知道其中的大多数快捷键。下面是所有在Microsoft Excel中的主要快捷键列表。

- **Ctrl + A :** 选择工作表中的所有内容。
- **Ctrl + B :** 粗体突出显示的选择。
- **Ctrl + I :** 斜体突出显示的选项。
- **Ctrl + K :** 插入链接。
- **Ctrl + U :** 强调突出显示的选择。
- **Ctrl + 1 :** 更改所选单元格的格式。
- **Ctrl + 5 :** 删除线高亮的选择。
- **Ctrl + P :** 调出打印对话框开始打印。
- **Ctrl + Z :** 撤消上一个操作。
- **Ctrl + F3 :** 打开Excel名称管理器。
- **Ctrl + F9:** 最小化当前窗口。
- **Ctrl + F10 :** 当前最大化选定的窗口。
- **Ctrl + F6 :** 打开的工作簿或窗口之间切换。
- **Ctrl + Page up :** 在同一个Excel文档中的Excel工作表之间移动。
- **Ctrl + Page down :** 在同一个Excel文档中的Excel工作表之间移动。
- **Ctrl + Tab :** 在两个或多个打开的Excel文件之间移动。
- **Alt + = :** 创建一个公式来相加所有上述单元格
- **Ctrl + ' :** 插入上述单元的当前所选的值进单元格。
- **Ctrl + Shift + ! :** 在逗号格式的格式号。
- **Ctrl + Shift + $:** 在货币格式格式号。
- **Ctrl + Shift + # :** 日期格式的格式号。

- **Ctrl + Shift + % :** 以百分比的格式格式号。
- **Ctrl + Shift + ^ :** 科学格式的格式号。
- **Ctrl + Shift + @ :** 在时间格式格式的数字。
- **Ctrl + Arrow key :** 移至文本下一节。
- **Ctrl + Space :** 选择整列。
- **Shift + Space :** 选择整行。
- **Ctrl + - :** 删除选定列或行。
- **Ctrl + Shift + = :** 插入新行或列。
- **Ctrl + Home :** 移动到单元格A1。
- **Ctrl + ~ :** 显示Excel公式或它们单元格值之间切换。
- **F2 :** 编辑选中的单元格。
- **F3 :** 经过一个名字已经创建F3将粘贴名称。
- **F4 :** 重复上一个动作。例如，如果你改变了文本的颜色在另一个单元格按F4会改变单元格相同颜色的文本。
- **F5 :** 转至所需的特定单元格。例如，C6。
- **F7 :** 拼写检查所选文本或文档。
- **F11 :** 从选择的数据创建图表。
- **Ctrl + Shift + ; :** 输入当前时间。
- **Ctrl + ; :** 输入当前日期。
- **Alt + Shift + F1 :** 插入新工作表。
- **Alt + Enter :** 当键入文本在单元格中按下Alt+ Enter键将移到下一行允许在一个单元格多行文本。
- **Shift + F3 :** 打开Excel公式窗口。
- **Shift + F5 :** 打开搜索框。

LinQ教程

世界各地的开发者总是遇到查询的问题，因为缺乏一个定义的路径的数据，并需要掌握的技术，如SQL，Web服务的XQuery等。

在Visual Studio 2008中引入，由Anders Hejlsberg设计LINQ（语言集成查询）允许编写查询，即使没有查询语言，如SQL，XML等知识 LINQ查询，可以由不同的数据类型来写。

LINQ查询示例

C

```
using System;
using System.Linq;

class Program
{
  static void Main()
  {
     string[] words = {"hello", "wonderful", "LINQ", "beautiful", "world"};
       //Get only short words
     var shortWords = from word in words
                      where word.Length <= 5
                      select word;

     //Print each word out
     foreach (var word in shortWords)
     {
         Console.WriteLine(word);
     }
     Console.ReadLine();
  }
}
```

VB

```
Module Module1
  Sub Main()
     Dim words As String() = {"hello", "wonderful", "LINQ", "beautiful", "world"}

     ' Get only short words
     Dim shortWords = From word In words _
                      Where word.Length <= 5 _
                      Select word

     ' Print each word out.
     For Each word In shortWords
        Console.WriteLine(word)
     Next
     Console.ReadLine()
  End Sub
End Module
```

当C＃或VB将上述代码被编译和执行时，它产生了以下结果：

```
hello
LINQ
world
```

LINQ的语法

LINQ有两种语法。这些是以下物质。

- Lamda (方法) 语法

  示例

  ```
  var longWords = words.Where( w =&gt; w.length &gt; 10);
  Dim longWords = words.Where(Function(w) w.length &gt; 10)
  ```

- Query (理解) 语法

  示例

  ```
  var longwords = from w in words where w.length &gt; 10;
  Dim longwords = from w in words where w.length &gt; 10
  ```

LINQ的类型

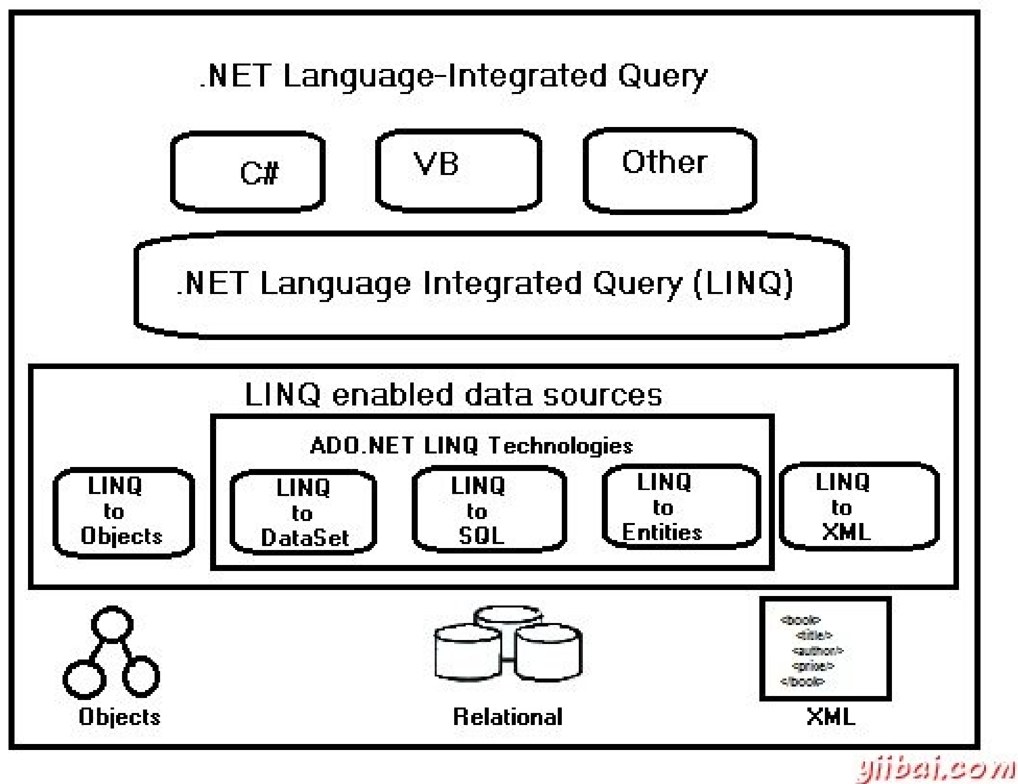
LINQ的类型在下面简要提及。

- LINQ 到 Objects
- LINQ 到XML(XLINQ)
- LINQ 到 DataSet
- LINQ 到 SQL (DLINQ)
- LINQ 到 Entities

除上述外，还有一个名为PLINQ一个LINQ类型，这是微软并行LINQ。

在.NETLINQ体系结构

LINQ有3层架构，其中最上层是由语言扩展和底层组成，通常对象实现了IEnumerable<T>或IQueryable的<T>泛型接口的数据源。该体系结构如下图。



查询表达式

查询表达式不过是一个LINQ查询，表示类似于SQL的查询使用操作符，如select，Where 和OrderBy的一种形式。查询表达式通常开始以关键字“From”。

访问标准的LINQ查询操作符，命名空间System.Query默认情况下应导入。这些表达式都写在C＃3.0声明性查询语法。

下面是一个例子来说明它由数据源创建一个完整的查询操作，查询表达式定义和查询执行。

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Operators
{
  class LINQQueryExpressions
  {
     static void Main()
     {
        // Specify the data source.
        int[] scores = new int[] { 97, 92, 81, 60 };

        // Define the query expression.
        IEnumerable<int> scoreQuery = from score in scores
                                      where score > 80
                                      select score;

           // Execute the query.
           foreach (int i in scoreQuery)
           {
              Console.Write(i + " ");
           }
           Console.ReadLine();
     }
  }
}
```

当上述代码被编译和执行时，它产生了以下结果：

```
97 92 81
```

扩展方法

引入.NET3.5，扩展方法只有静态类中声明，并允许包含自定义方法的对象进行一些精确的查询操作来扩展类，而不由这个类的实际成员。这些也可以被重载。

简而言之，扩展方法被用来转换查询表达式为传统的方法调用（面向对象）。

LINQ和存储过程的区别

还有就是LINQ和存储过程之间存在差异的数组。这些差别将在下面提及。

- 存储过程比LINQ查询速度更快，因为它们遵循预期的执行计划。
- 在比较执行LINQ查询这很容易避免，而不是存储过程作为前者在编译时Visual Studio的智能提示支持，以及全类型检查运行时错误。
- LINQ允许调试通过使用.NET调试器不是在存储过程。
- LINQ提供了相对于存储过程，其中有必要重新写为数据库类型多样的代码的多个数据库的支持。

- LINQ基于解决方案的部署是容易和简单，相比部署一套存储过程。

LINQ的需要

在此之前LINQ，有必要学习C#，SQL，和各种API结合在一起既要形成一个完整的应用程序。因为，这些数据源和编程语言面临的阻抗不匹配;需要短编码的感觉。

下面是在查询数据的LINQ来临之前有多少不同的技术中使用由开发的一个例子。

```
SqlConnection sqlConnection = new SqlConnection(connectString);
SqlConnection.Open();
System.Data.SqlClient.SqlCommand sqlCommand = new SqlCommand();
sqlCommand.Connection = sqlConnection;
sqlCommand.CommandText = "Select * from Customer";
return sqlCommand.ExecuteReader (CommandBehavior.CloseConnection)
```

有趣的是，出了特征代码行，查询获取只有最后两个定义。使用LINQ，相同的数据的查询可以写成一个可读颜色编码形式象，如下面那太在很短提下列到之一。

```
Northwind db = new Northwind(@"C:\Data\Northwnd.mdf");
var query = from c in db.Customers
            select c;
```

LINQ的优点

LINQ提供了一系列的优势，其中最重要的是其强大的表达能力，使开发表达声明。一些LINQ的优点如下。

- LINQ提供语法高亮，证明有助于找出在设计时的错误。
- LINQ提供智能感知这意味着很容易写更精确的查询。
- 写LINQ代码是相当快的，因此开发时间也被显著减少。
- LINQ使得调试方便，因为它在C#语言的集成。
- 两个表之间的关系看很容易使用LINQ由于其分层特征，这使得编写查询在更短的时间加入多个表。
- LINQ允许一个单一的LINQ语法的使用，同时查询多个不同的数据源，这是主要是因为其统一的基础。
- LINQ是可扩展的，这意味着有可能使用LINQ的知识来查询新的数据源类型。
- LINQ提供了一个查询连接多个数据源，以及突破复杂问题转换为一组短的查询易于调试的工具。

- LINQ提供易于改造转换一种数据类型到另一种，如SQL数据转换为XML数据。

LINQ环境安装设置 - LinQ教程

开始LINQ程序之前，最好先了解设立LINQ环境的细微差异。 LINQ需要.NET框架，一种革命性的平台有不同类型的应用程序。 LINQ查询可以写成无论是在C#或Visual Basic都很方便。

微软提供了这两种语言的工具，即在Visual Studio中编写的C#和Visual Basic。我们的例子都是编译并写在Visual Studio 2010中；但是，Visual Basic2013版的也可以使用。这是最新版本，并有许多相似之处与Visual Studio2012。

在Windows7安装Visual Studio 2010

Visual Studio可以从像DVD安装介质进行安装。需要管理员权限才能成功地在系统上安装Visual Basic2010。重要的是如果断开所有可移动USB否则安装可能会失败。一些硬件要求具有用于安装必需如下所述。

硬件要求：

- 1.6 GHz 或 更高
- 1 GB RAM
- 3 GB(可用硬盘空间)
- 5400 RPM 硬盘驱动器
- DirectX 9 兼容的视频卡
- DVD-ROM 驱动

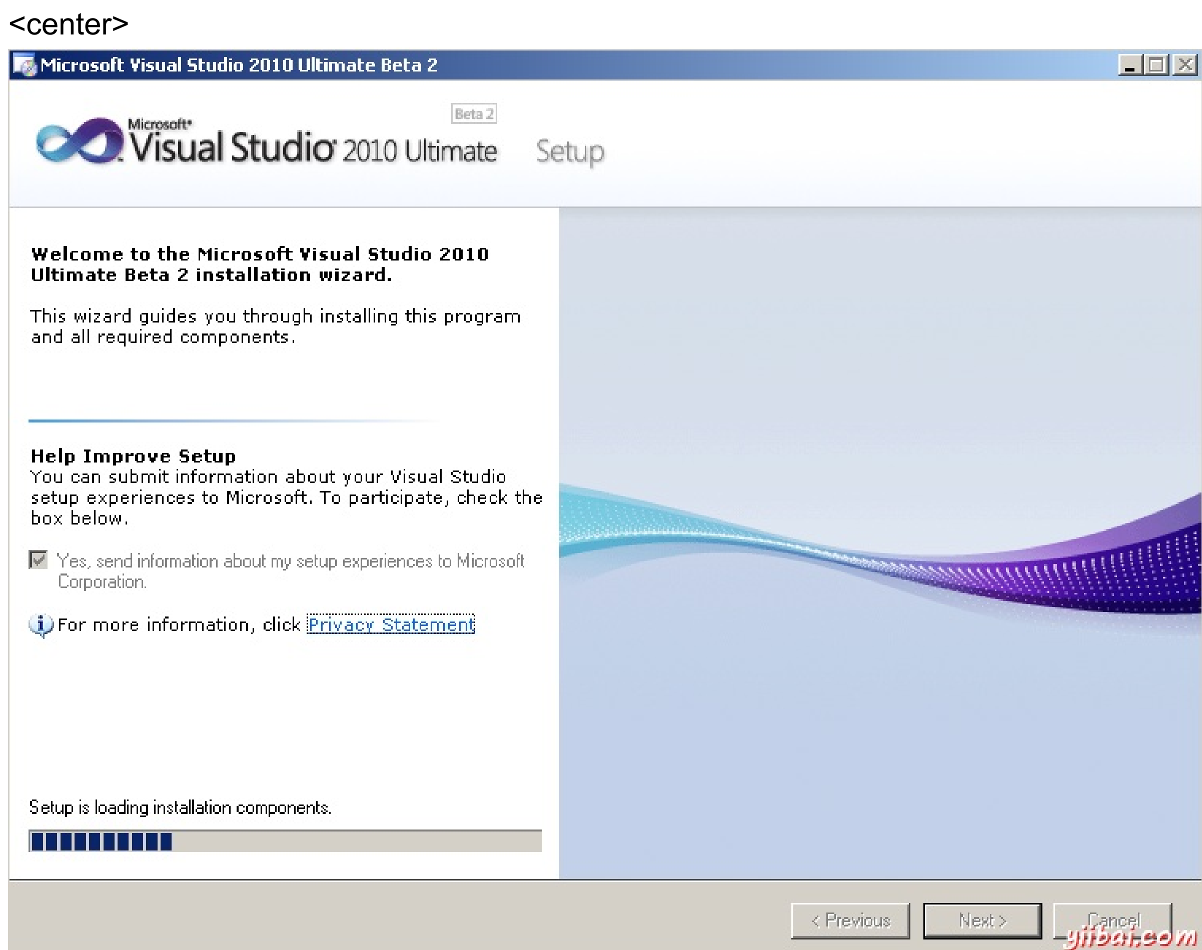
安装步骤：

- 步骤 1. 首先使用Visual Studio2010包，在插入DVD后，点击从媒体出现在屏幕上的弹出框安装或运行程序。
- 步骤 2.现在设置了Visual Studio后会出现在屏幕上。选择 Install Microsoft Visual Studio 2010.

<center>


</center>

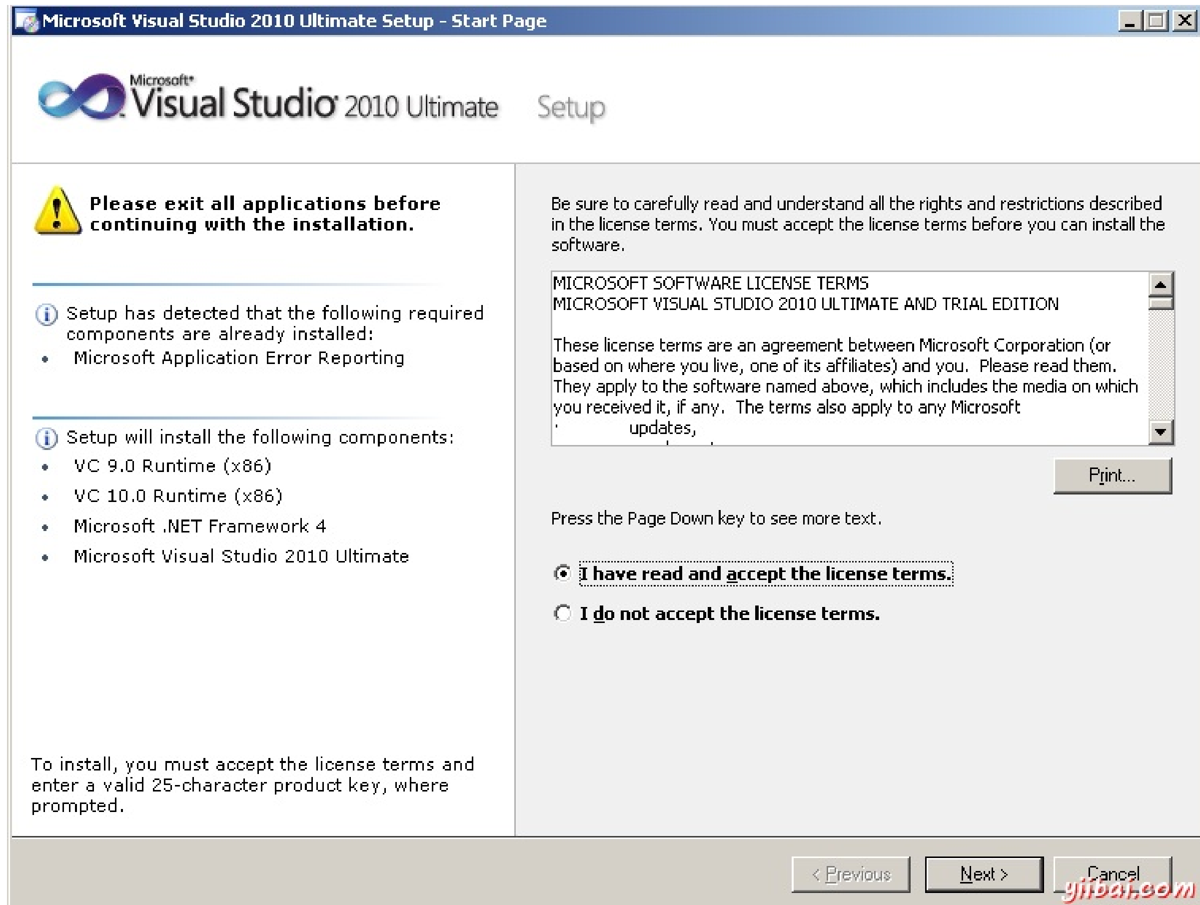
- 步骤 3. 只要点击，它的进程将得到发起和设立窗口将出现在屏幕上。加载安装组件，将需要一些时间完成后，单击 Next 按钮进入下一步。

<center>


</center>

- 步骤 4. 这是安装的最后一步，并开始页面将出现在其中只是选择“我已阅读并接受许可条款”，然后单击Next按钮。

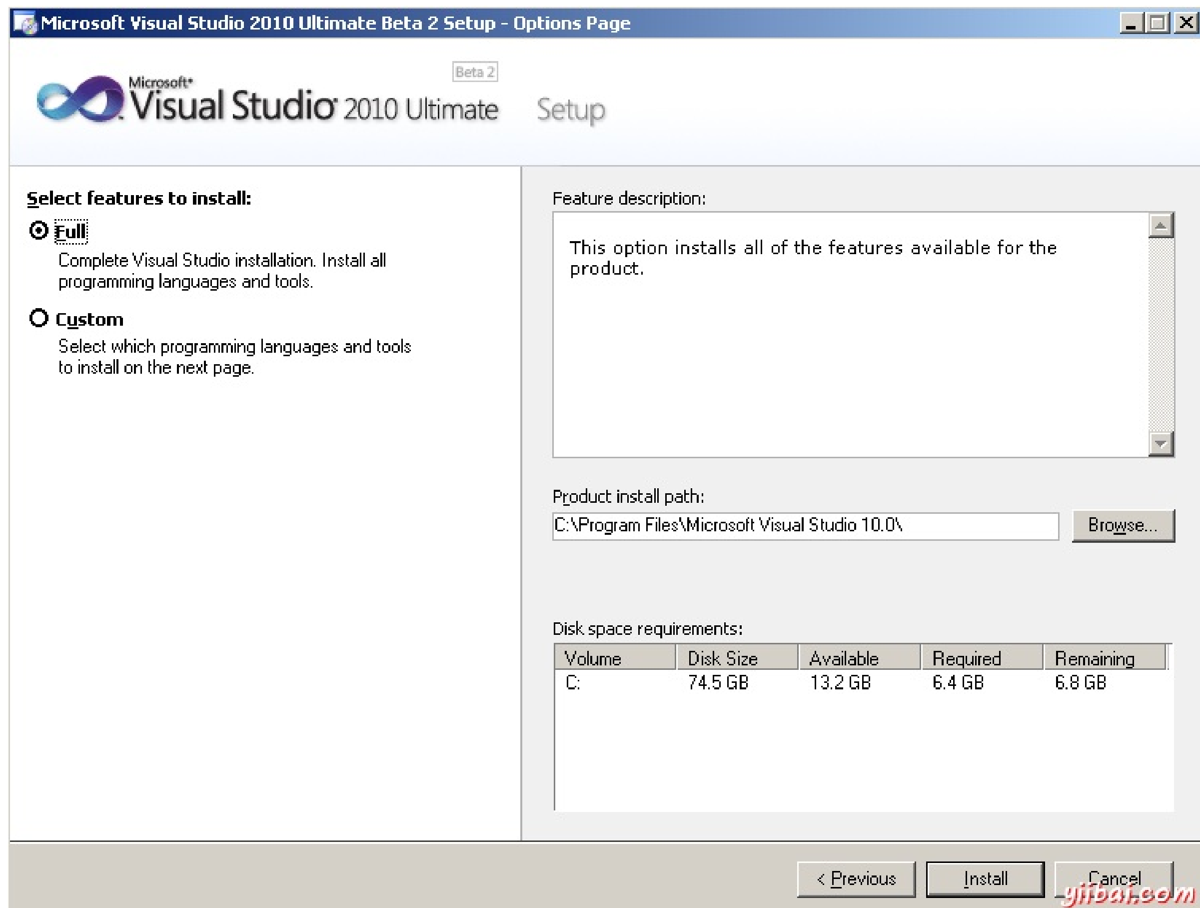
<center>



</center>

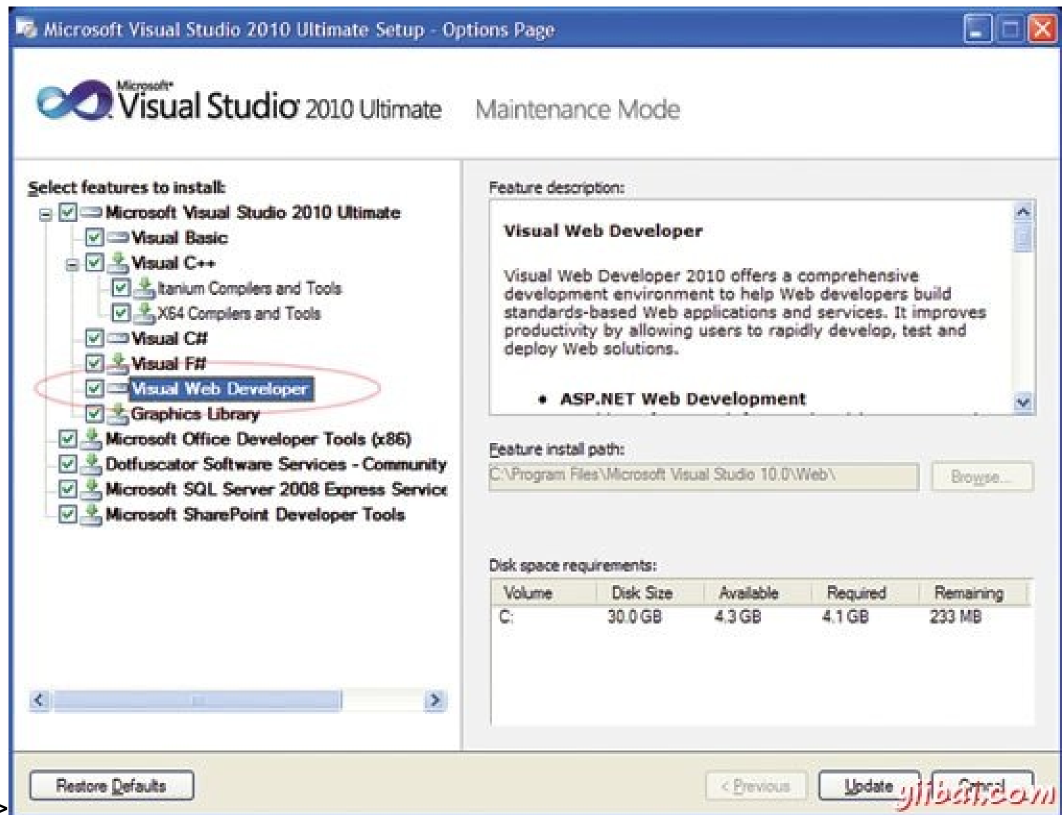
- 步骤 5.现在选择功能出现在屏幕上的安装选项。您可以选择Full 或 Custom选项。如果你有比磁盘空间要求所需的显示更少的磁盘空间，那么请定制安装。

<center>



</center>

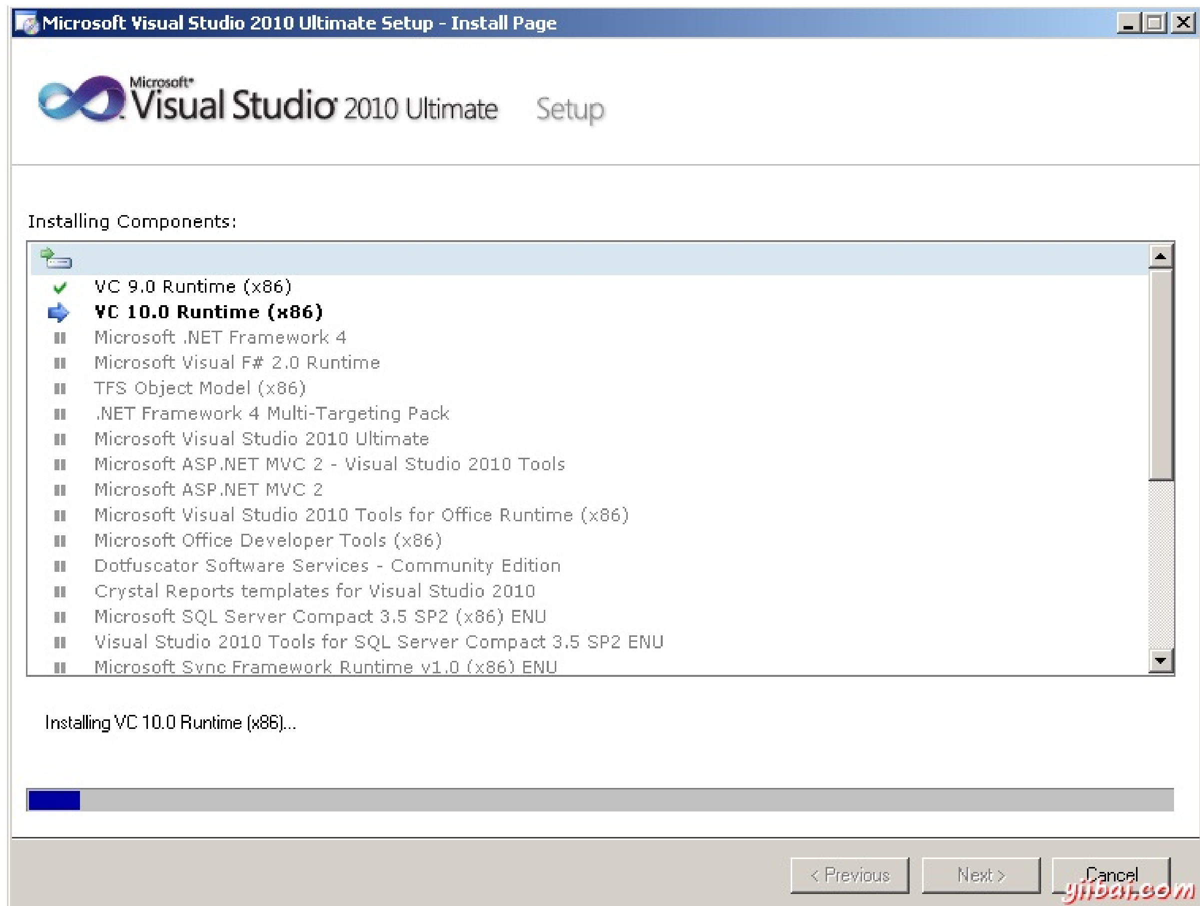
- Step 6.当选择自定义选项，会出现下面的窗口。选择想要安装，点击更新或者前往第7步；建议不要去自定义选项，因为在未来，你可能需要选择了不具备的功能。



<center>
</center>

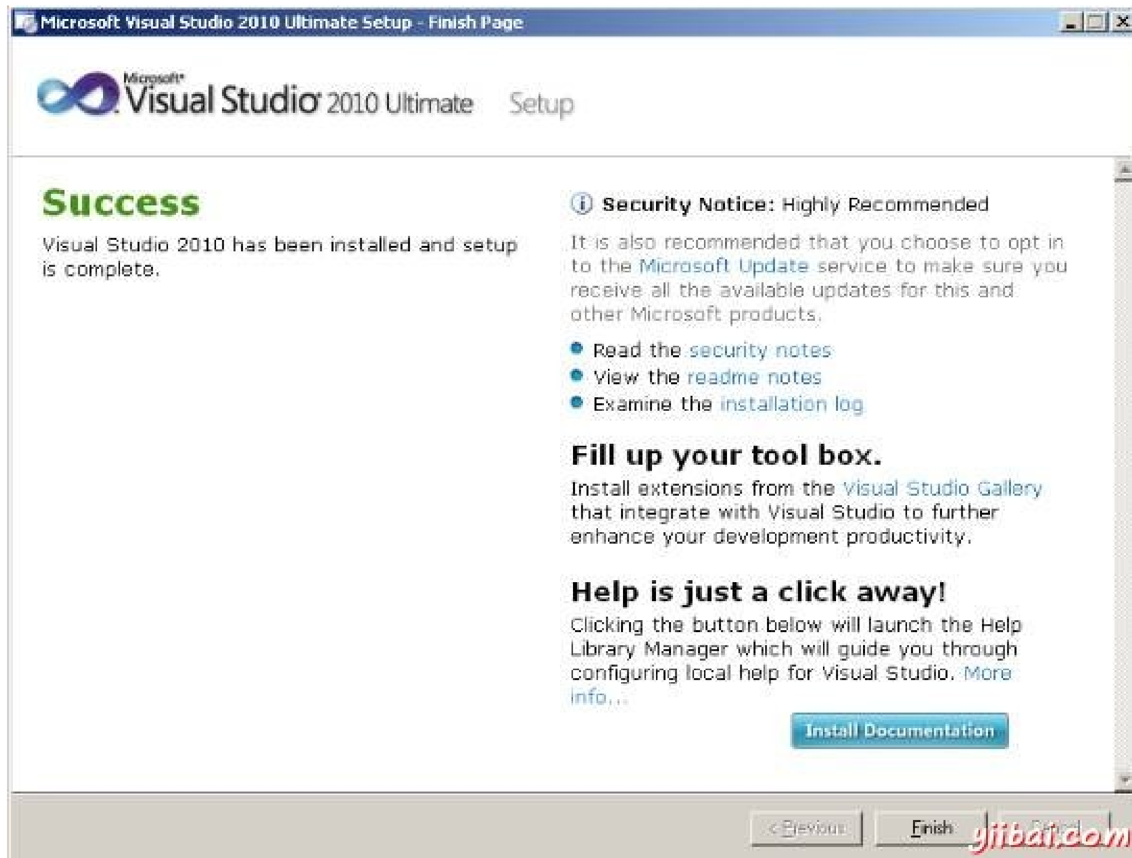
不久，一个弹出窗口会显示并安装将启动这可能需要很长的时间。请记住，这是要安装了所有组件。

<center>



</center>

最后，能够在安装已成功完成一个窗口查看一条消息。

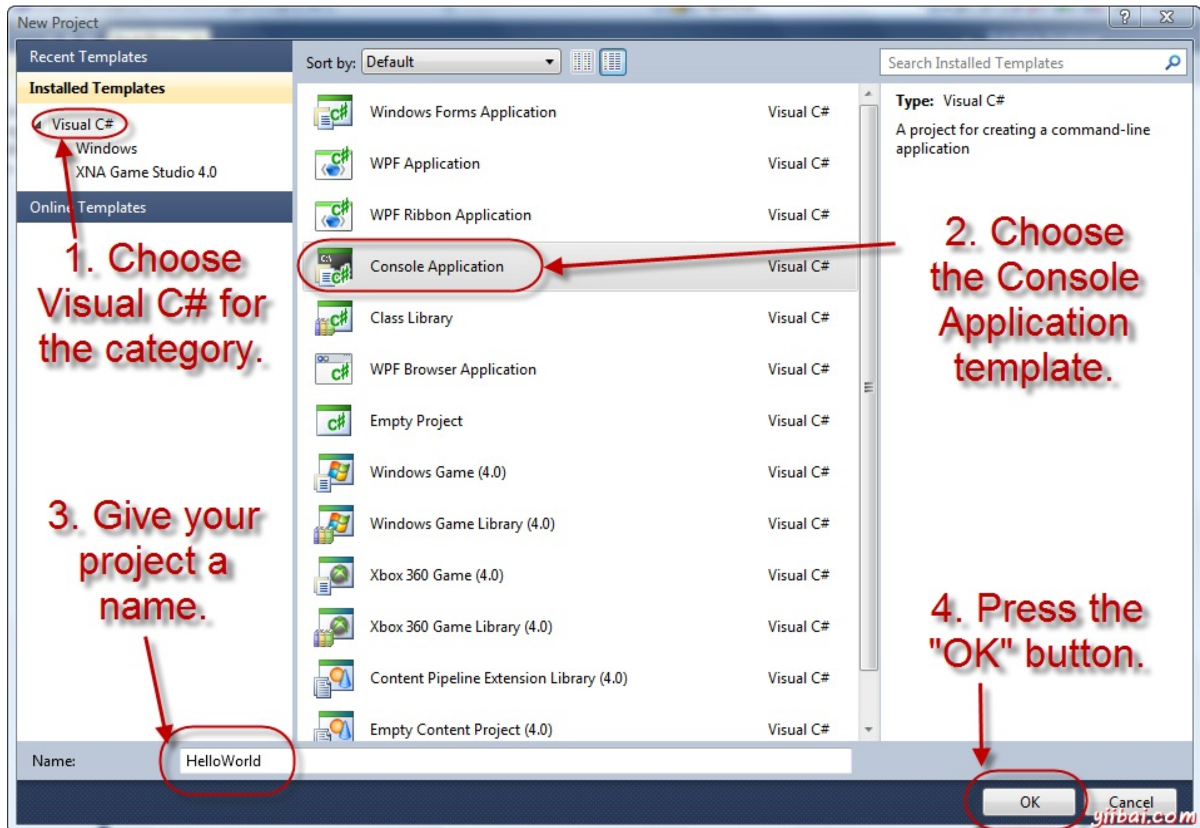
<center>


</center>

在Visual Studio 2010中使用C#写LINQ程序

- 1. 启动Visual Studio2010终极版，然后从菜单选择文件后新建项目。
- 2. 一个新的项目对话框会出现在屏幕上。
- 3. 现在选择的Visual C#作为已安装的模板下的一个类别，下一个选择控制台应用程序模板，如下图所示。

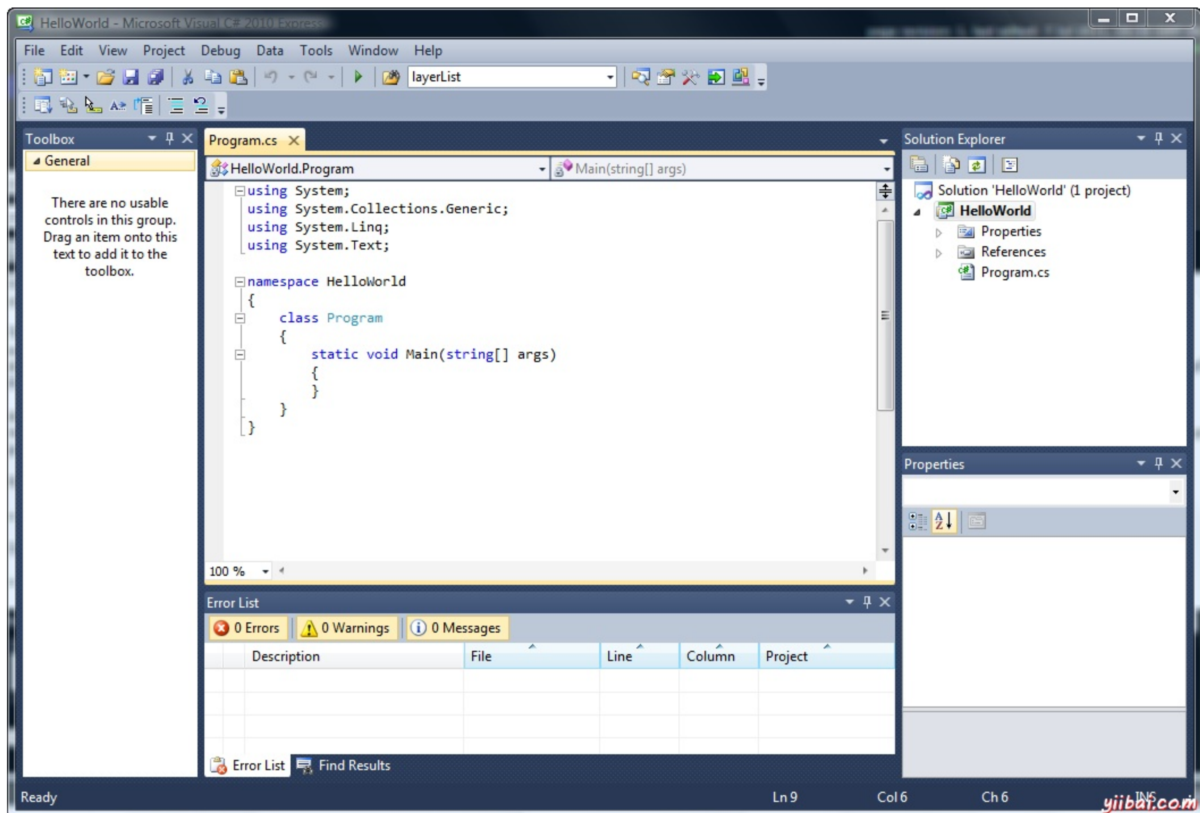
<center>



</center>

- 4. 提供一个名称，在底部的名称框，然后点击 OK。
- 5. 新项目将出现在解决方案资源管理器中的一个新的对话框屏幕上的右侧。

<center>



</center>

- 6. 现在，从解决方案资源管理器中选择Program.cs中，您可以在其中启动编辑器窗口查看代码使用 ‘using System’.
- 7. 在这里，可以开始编写下面的C#程序。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HelloWorld
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Hello World")
         Console.ReadKey();
      }
   }
}
```

- 8. 按F5键运行项目。强烈建议保存项目通过选择文件>全部保存在运行项目之前。

在Visual Studio 2010中使用LINQ 写VB程序

- 1. 启动Visual Studio2010终极版，然后选择文件后新建项目从菜单。

- 2.新项目对话框将出现在屏幕上。
- 3. 现在，选择Visual Basic中已安装的模板，下一个选择控制台应用程序模板，下一个类别。
- 4. 提供一个名称，在底部的名称框，然后按OK。
- 5. 得到Module1.vb 如屏幕中内容。从这里开始使用LINQ 编写VB代码。

```
Module Module1
  Sub Main()
     Console.WriteLine("Hello World")
     Console.ReadLine()
  End Sub
End Module
```

- 6. 按F5键运行项目。强烈建议保存项目通过选择文件>全部保存在运行项目之前。

当C3或VB上面的代码被编译和运行，它会产生以下结果：

LINQ投影操作 - LinQ教程

投影是在其中一个对象被变换成仅特定性能一种全新的形式的操作。

运算符	描述	C#查询表达式语法	VB查询表达式语法
Select	操作转换函数的基础项目值	select	Select
SelectMany	操作项目的值是根据上的转换函数，以及拼合成一个单一的序列的序列	使用多个from子句	使用多个from子句

Select的例子- 查询表达式

C

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Operators
{
  class Program
  {
     static void Main(string[] args)
     {
        List<string> words = new List<string>() { "an", "apple", "a", "day" };

        var query = from word in words
                    select word.Substring(0, 1);

        foreach (string s in query)
           Console.WriteLine(s);
           Console.ReadLine();
     }
  }
}
```

VB

```
Module Module1
  Sub Main()
     Dim words = New List(Of String) From {"an", "apple", "a", "day"}

     Dim query = From word In words
                 Select word.Substring(0, 1)

     Dim sb As New System.Text.StringBuilder()
     For Each letter As String In query
        sb.AppendLine(letter)
        Console.WriteLine(letter)
     Next
        Console.ReadLine()
  End Sub
End Module
```

当在C＃或VB上面的代码被编译和执行时，它产生以下结果：

```
a
a
a
d
```

SelectMany例子- 查询表达式

C

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Operators
{
  class Program
  {
     static void Main(string[] args)
     {
        List<string> phrases = new List<string>() { "an apple a day", "the quick brown fo

        var query = from phrase in phrases
                    from word in phrase.Split(' ')
                    select word;

        foreach (string s in query)
           Console.WriteLine(s);
           Console.ReadLine();
     }
  }
}
```

VB

```
Module Module1
  Sub Main()
     Dim phrases = New List(Of String) From {"an apple a day", "the quick brown fox"}

     Dim query = From phrase In phrases
                 From word In phrase.Split(" "c)
                 Select word

     Dim sb As New System.Text.StringBuilder()
     For Each str As String In query
        sb.AppendLine(str)
        Console.WriteLine(str)
     Next
        Console.ReadLine()
  End Sub
End Module
```

当在C＃或VB上面的代码被编译和执行时，它产生了以下结果：

```
an
apple
a
day
the
quick
brown
fox
```

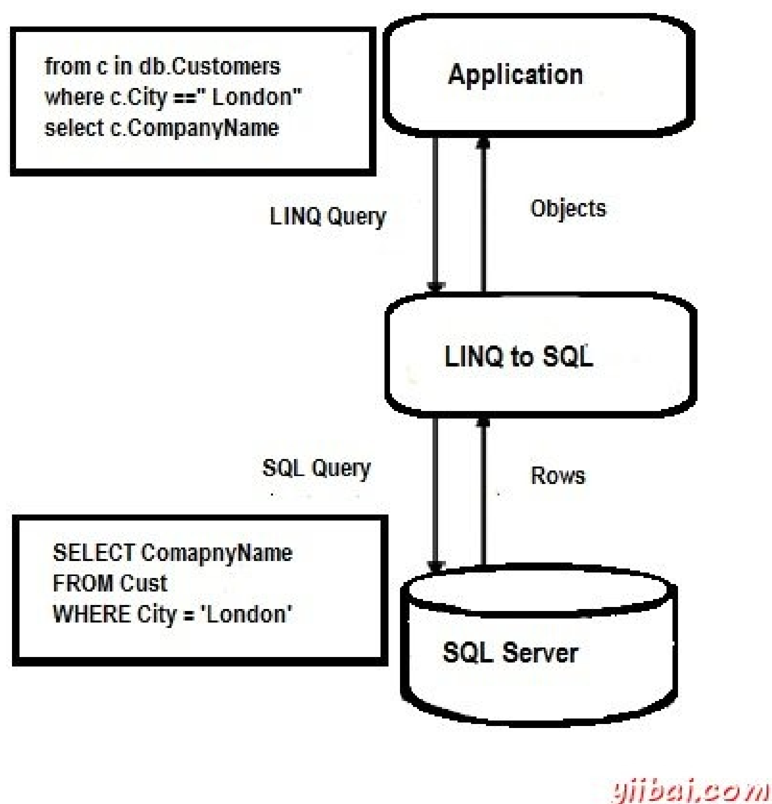
LINQ SQL - LinQ教程

LINQ为SQL提供了一个基础结构(运行时间)，关系数据作为对象的管理。 这是3.5版本的.NET框架的一个组成部分，并巧妙地做对象模型到SQL的语言集成查询的转换。这些查询被发送给数据库来执行。从数据库中获得的结果后，LINQ到SQL再次它们转换为对象。

LINQ到SQL简介

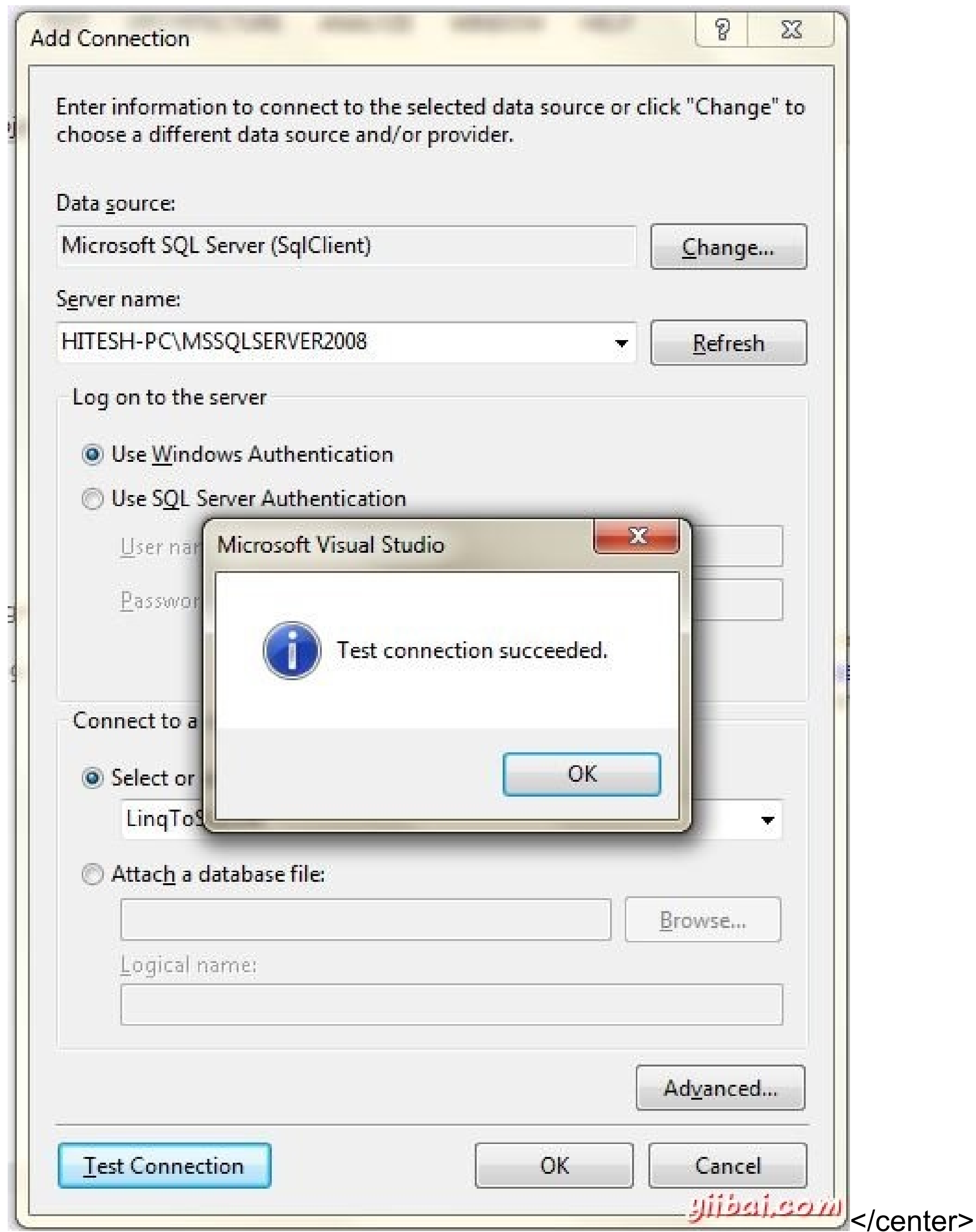
对于大多数ASP.NET开发，LINQ到SQL（也称为DLINQ）是Langauage集成查询通用部分，因为这允许在SQL服务器器数据库查询数据通过使用常规的LINQ表达式。它还允许更新，删除和插入数据，但它受到的唯一缺点是它受SQL服务器数据库的限制。 但是，也有LINQ的许多好处，SQL在ADO.NET一样降低了复杂性，编码等等。

下面是表示LINQ执行架构到SQL的示意图。



如何使用LINQ到SQL?

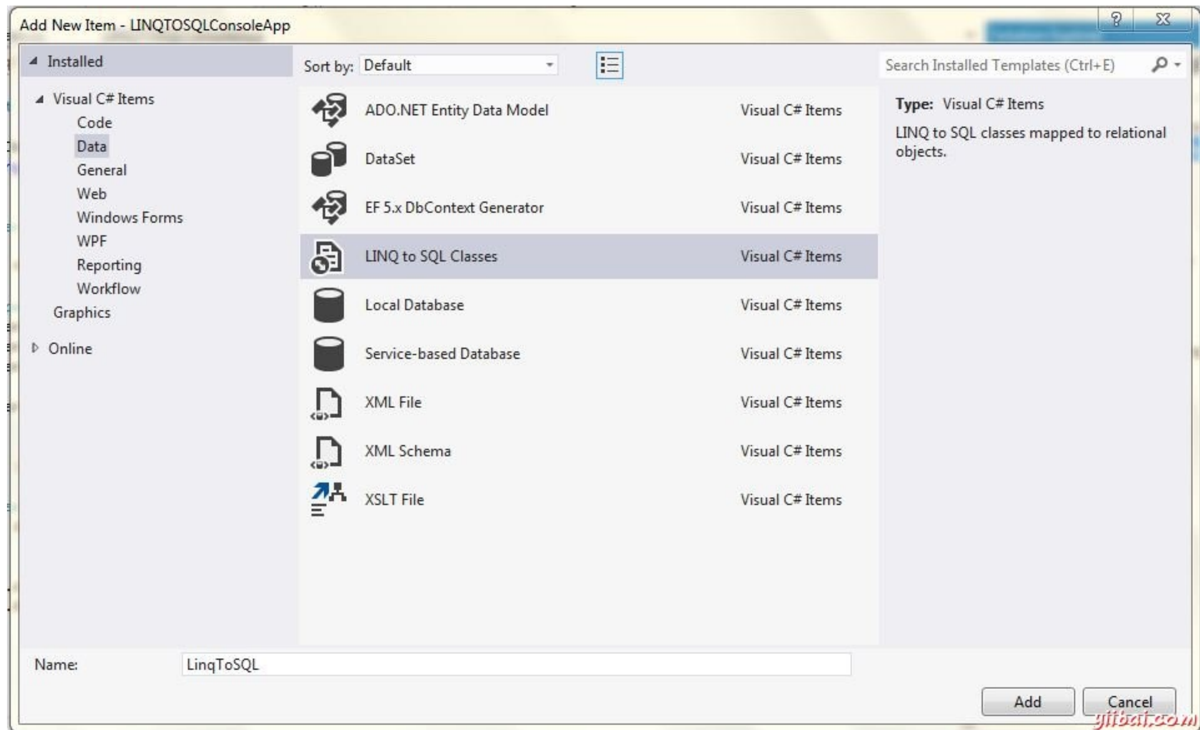
- 步骤 1: 创建一个新的“数据连接”到数据库服务器。View -> Server Explorer -> Data Connections -> Add Connection

<center style="box-sizing: border-box;">


</center>

- 步骤2: 添加LINQ 到 SQL类文件

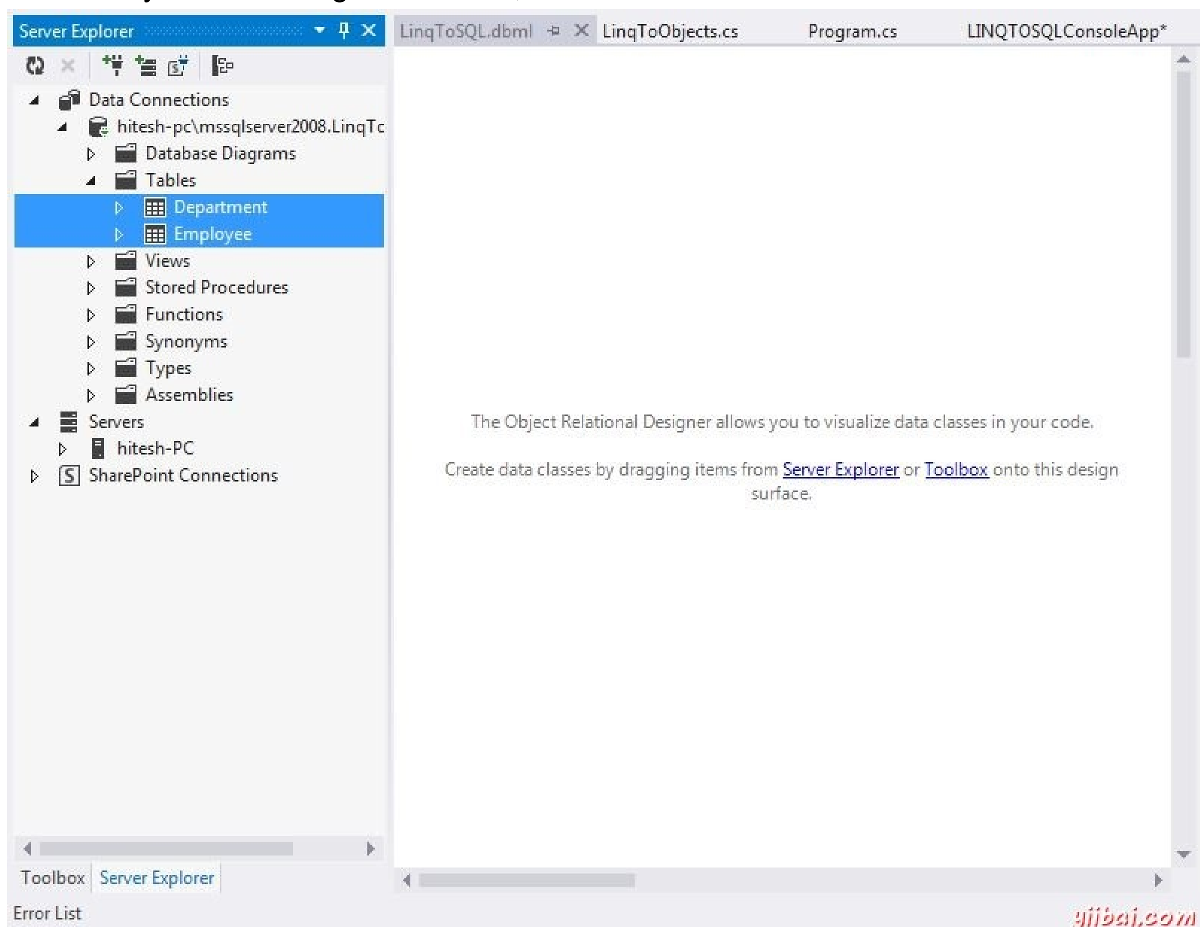
<center style="box-sizing: border-box;">



</center>

- 步骤 3: 选择数据库并拖动表格拖放到新的LINQ 到 SQL类文件。

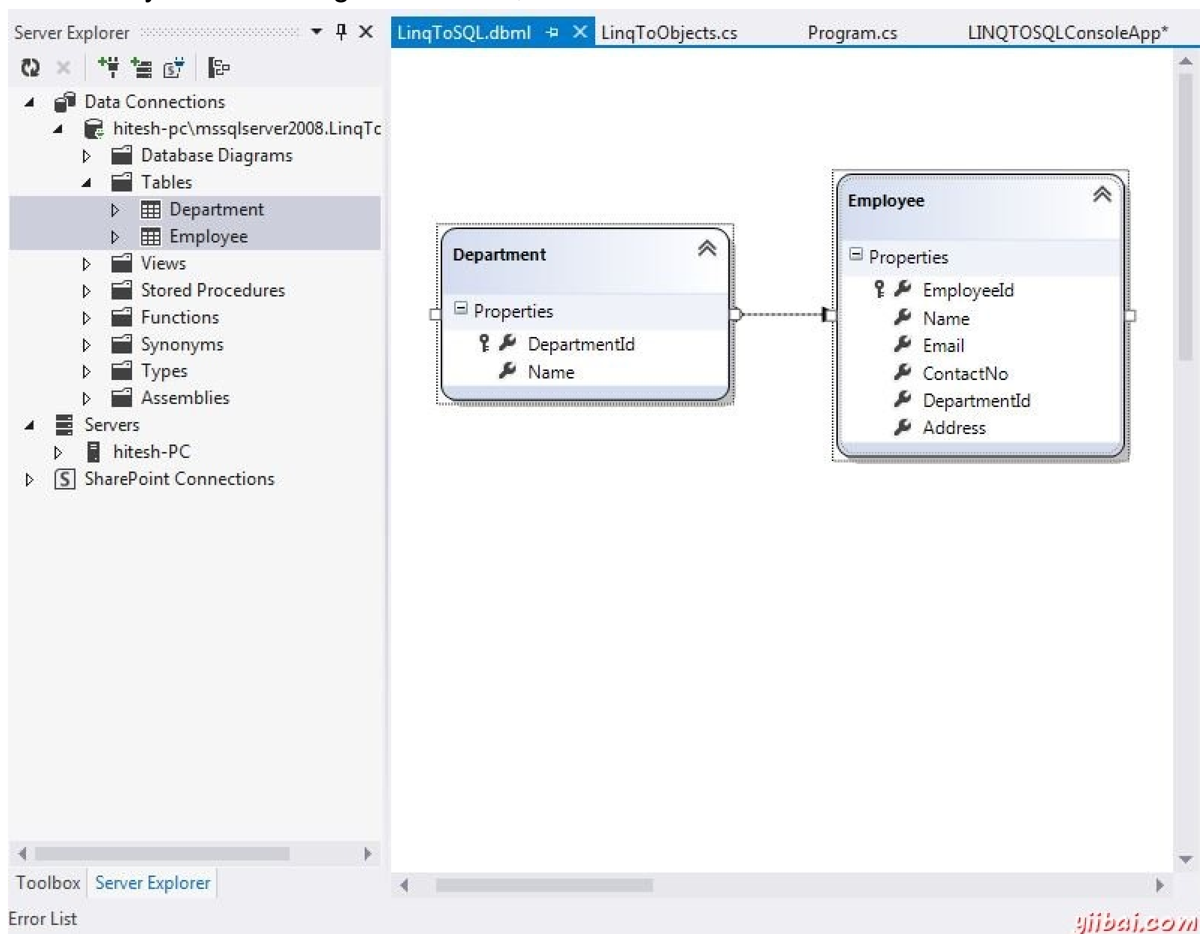
<center style="box-sizing: border-box;">



</center>

- 步骤 4: 添加表类文件。

  <center style="box-sizing: border-box;">



  </center>

使用LINQ到SQL查询

执行一个查询使用LINQ到SQL规则类似到一个标准的LINQ查询，即 执行查询或者延迟或马上执行。 在执行使用LINQ到SQL查询的发挥作用，这些都是下述的组件。

- **LINQ到SQL API** – 请求查询执行代表一个应用程序，并把它交给LINQ到SQL提供程序
- **LINQ 到SQL提供程序** - 转换查询到Transact SQL（T-SQL），并发送新的查询到ADO提供程序执行
- **ADO** 提供程序 - 执行查询之后，发送结果在一个DataReader形式的LINQ 到 SQL，提供这反过来将其转换成用户的对象的形式

应当指出的是，在进行一个LINQ到SQL查询之前，重要的是要连接到经由DataContext类的数据源。

使用LINQ 到SQL插入，更新和删除

添加或插入

C

```
using System;
using System.Linq;

namespace LINQtoSQL
{
  class LinqToSQLCRUD
  {
     static void Main(string[] args)
     {
        string connectString = System.Configuration.ConfigurationManager.ConnectionString

        LinqToSQLDataContext db = new LinqToSQLDataContext(connectString);

        //Create new Employee
        Employee newEmployee = new Employee();
        newEmployee.Name = "Michael";
        newEmployee.Email = "yourname@companyname.com";
        newEmployee.ContactNo = "343434343";
        newEmployee.DepartmentId = 3;
        newEmployee.Address = "Michael - USA";

        //Add new Employee to database
        db.Employees.InsertOnSubmit(newEmployee);

        //Save changes to Database.
        db.SubmitChanges();

        //Get new Inserted Employee
        Employee insertedEmployee = db.Employees.FirstOrDefault(e =>e.Name.Equals("Michae

        Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3},
                          insertedEmployee.EmployeeId, insertedEmployee.Name, insertedEmp
                          insertedEmployee.ContactNo, insertedEmployee.Address);

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Module Module1
  Sub Main()
     Dim connectString As String = System.Configuration.ConfigurationManager.ConnectionSt

     Dim db As New LinqToSQLDataContext(connectString)

     Dim newEmployee As New Employee()
     newEmployee.Name = "Michael"
     newEmployee.Email = "yourname@companyname.com"
     newEmployee.ContactNo = "343434343"
     newEmployee.DepartmentId = 3
     newEmployee.Address = "Michael - USA"

     db.Employees.InsertOnSubmit(newEmployee)

     db.SubmitChanges()

     Dim insertedEmployee As Employee = db.Employees.FirstOrDefault(Function(e) e.Name.Eq

     Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3}, Add

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub
End Module
```

当C#或VB上面的代码被编译并运行，它会产生以下结果：

```
Emplyee ID = 4, Name = Michael, Email = yourname@companyname.com, ContactNo =
343434343, Address = Michael - USA

Press any key to continue.
```

更新

C

```
using System;
using System.Linq;

namespace LINQtoSQL
{
  class LinqToSQLCRUD
  {
     static void Main(string[] args)
     {
        string connectString = System.Configuration.ConfigurationManager.ConnectionString

        LinqToSQLDataContext db = new LinqToSQLDataContext(connectString);

        //Get Employee for update
        Employee employee = db.Employees.FirstOrDefault(e =>e.Name.Equals("Michael"));

        employee.Name = "George Michael";
        employee.Email = "yourname@companyname.com";
        employee.ContactNo = "99999999";
        employee.DepartmentId = 2;
        employee.Address = "Michael George - UK";

        //Save changes to Database.
        db.SubmitChanges();

        //Get Updated Employee
        Employee updatedEmployee = db.Employees.FirstOrDefault(e =>e.Name.Equals("George

        Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3},
                          updatedEmployee.EmployeeId, updatedEmployee.Name, updatedEmploy
                          updatedEmployee.ContactNo, updatedEmployee.Address);

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Module Module1
  Sub Main()
     Dim connectString As String = System.Configuration.ConfigurationManager.ConnectionSt

     Dim db As New LinqToSQLDataContext(connectString)

     Dim employee As Employee = db.Employees.FirstOrDefault(Function(e) e.Name.Equals("Mi

     employee.Name = "George Michael"
     employee.Email = "yourname@companyname.com"
     employee.ContactNo = "99999999"
     employee.DepartmentId = 2
     employee.Address = "Michael George - UK"

     db.SubmitChanges()

     Dim updatedEmployee As Employee = db.Employees.FirstOrDefault(Function(e) e.Name.Equ

     Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3}, Add

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub
End Module
```

当C#或VB上面的代码被编译并运行，它会产生以下结果：

```
Emplyee ID = 4, Name = George Michael, Email = yourname@companyname.com, ContactNo =
999999999, Address = Michael George - UK

Press any key to continue.
```

删除

C

```
using System;
using System.Linq;

namespace LINQtoSQL
{
  class LinqToSQLCRUD
  {
     static void Main(string[] args)
     {
        string connectString = System.Configuration.ConfigurationManager.ConnectionString

        LinqToSQLDataContext db = newLinqToSQLDataContext(connectString);

        //Get Employee to Delete
        Employee deleteEmployee = db.Employees.FirstOrDefault(e =>e.Name.Equals("George M

        //Delete Employee
        db.Employees.DeleteOnSubmit(deleteEmployee);

        //Save changes to Database.
        db.SubmitChanges();

        //Get All Employee from Database
        var employeeList = db.Employees;
        foreach (Employee employee in employeeList)
        {
           Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3
                             employee.EmployeeId, employee.Name, employee.Email, employee
        }

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Module Module1
  Sub Main()
     Dim connectString As String = System.Configuration.ConfigurationManager.ConnectionSt

     Dim db As New LinqToSQLDataContext(connectString)

     Dim deleteEmployee As Employee = db.Employees.FirstOrDefault(Function(e) e.Name.Equa

     db.Employees.DeleteOnSubmit(deleteEmployee)

     db.SubmitChanges()

     Dim employeeList = db.Employees
     For Each employee As Employee In employeeList
        Console.WriteLine("Employee Id = {0} , Name = {1}, Email = {2}, ContactNo = {3}",
     Next

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub
End Module
```

当C#或VB上面的代码被编译并运行，它会产生以下结果：

```
Emplyee ID = 1, Name = William, Email = abc@gy.co, ContactNo = 999999999
Emplyee ID = 2, Name = Miley, Email = amp@esds.sds, ContactNo = 999999999
Emplyee ID = 3, Name = Benjamin, Email = asdsad@asdsa.dsd, ContactNo =

Press any key to continue.
```

LINQ对象 - LinQ教程

LINQ到Objects提供任何LINQ查询支持的IEnumerable<T>访问内存中的数据集合，而不需要任何LINQ提供程序(API)的情况下，使用LINQ到SQL或LINQ到XML。

LINQ到对象简介

查询在LINQ到对象返回IEnumerable<T>只有类型的变量。 总之，LINQ到Objects在早期提供了一种新方法到集合，它需要写很长代码换成声明性代码的集合，清楚地描述了所需的数据编码(foreach循环复杂得多)进行数据检索至所需关键的检索。

LINQ有许多优势超过传统的foreach循环，更易读，强大的过滤，分组的能力，增强排序以最小的应用程序的编码对象。这样LINQ查询在性质上也更加紧凑，并且移植到任何其它数据源没有任何修改或只需稍加修改。

下面是一个简单的LINQ到对象的例子：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LINQtoObjects
{
  class Program
  {
     static void Main(string[] args)
     {
        string[] tools = { "Tablesaw", "Bandsaw", "Planer", "Jointer", "Drill",
                           "Sander" };
        var list = from t in tools
                   select t;

        StringBuilder sb = new StringBuilder();

        foreach (string s in list)
        {
           sb.Append(s + Environment.NewLine);
        }
        Console.WriteLine(sb.ToString(), "Tools");
        Console.ReadLine();
     }
  }
}
```

在这个例子中，字符串(工具)的阵列被用作对象的集合，使用LINQ到对象进行查询。

```
Objects query is:
var list = from t in tools
           select t;
```

当上述代码被编译和执行时，它产生了以下结果：

```
Tablesaw
Bandsaw
Planer
Jointer
Drill
Sander
```

使用LINQ到内存中的对象集合查询

C

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace LINQtoObjects
{
  class Department
  {
     public int DepartmentId { get; set; }
     public string Name { get; set; }
  }

  class LinqToObjects
  {
     static void Main(string[] args)
     {
        List<Department> departments = new List<Department>();
        departments.Add(new Department { DepartmentId = 1, Name = "Account" });
        departments.Add(new Department { DepartmentId = 2, Name = "Sales" });
        departments.Add(new Department { DepartmentId = 3, Name = "Marketing" });

        var departmentList = from d in departments
                             select d;

        foreach (var dept in departmentList)
        {
           Console.WriteLine("Department Id = {0} , Department Name = {1}",
                             dept.DepartmentId, dept.Name);
        }
        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Imports System.Collections.Generic
Imports System.Linq

Module Module1
  Sub Main(ByVal args As String())

     Dim account As New Department With {.Name = "Account", .DepartmentId = 1}
     Dim sales As New Department With {.Name = "Sales", .DepartmentId = 2}
     Dim marketing As New Department With {.Name = "Marketing", .DepartmentId = 3}

     Dim departments As New System.Collections.Generic.List(Of Department)(New Department

     Dim departmentList = From d In departments

     For Each dept In departmentList
        Console.WriteLine("Department Id = {0} , Department Name = {1}", dept.DepartmentI
     Next

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub

  Class Department
     Public Property Name As String
     Public Property DepartmentId As Integer
  End Class
End Module
```

当C＃或VB的上述代码被编译和执行时，它产生了以下结果：

```
Department Id = 1, Department Name = Account
Department Id = 2, Department Name = Sales
Department Id = 3, Department Name = Marketing

Press any key to continue.
```

LINQ Dataset（数据集） - LinQ教程

数据集提供了一个非常有用的数据表示在存储器中，用于数据的基础应用的一个不同范围。LINQ到数据集为一个LINQ来执行查询的数据集的数据无忧的方式ADO.NET工具的技术，并提高生产力。

LINQ到数据集的介绍

LINQ到数据集已取得查询简单的开发任务。 它们并不需要在一个特定的查询语言，可以用编程语言编写相同的查询。LINQ到数据集也是用于查询，其中数据从多个数据源合并使用。这也并不需要任何LINQ提供程序从内存中集合访问从LINQ 到 SQL和LINQ 到 XML读取数据。

下面是其中一个数据源首先获得，然后将数据集填充的两个数据表一个LINQ到数据集的查询的一个简单的例子。关系是表和LINQ查询被Join子句方式，对两个表创建在两者之间建立的。最后，foreach循环用于显示所期望的结果。

C

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LINQtoDataset
{
  class Program
  {
     static void Main(string[] args)
     {
        string connectString = System.Configuration.ConfigurationManager.ConnectionString

        string sqlSelect = "SELECT * FROM Department;" +
                           "SELECT * FROM Employee;";

        // Create the data adapter to retrieve data from the database
        SqlDataAdapter da = new SqlDataAdapter(sqlSelect, connectString);

        // Create table mappings
        da.TableMappings.Add("Table", "Department");
        da.TableMappings.Add("Table1", "Employee");

        // Create and fill the DataSet
        DataSet ds = new DataSet();
        da.Fill(ds);

        DataRelation dr = ds.Relations.Add("FK_Employee_Department",
                          ds.Tables["Department"].Columns["DepartmentId"],
                          ds.Tables["Employee"].Columns["DepartmentId"]);

        DataTable department = ds.Tables["Department"];
        DataTable employee = ds.Tables["Employee"];

        var query = from d in department.AsEnumerable()
                    join e in employee.AsEnumerable()
                    on d.Field<int>("DepartmentId") equals
                    e.Field<int>("DepartmentId")
                    select new
                    {
                       EmployeeId = e.Field<int>("EmployeeId"),
                       Name = e.Field<string>("Name"),
                       DepartmentId = d.Field<int>("DepartmentId"),
                       DepartmentName = d.Field<string>("Name")
                    };

        foreach (var q in query)
        {
           Console.WriteLine("Employee Id = {0} , Name = {1} , Department Name = {2}",
                             q.EmployeeId, q.Name, q.DepartmentName);
        }

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Imports System.Data.SqlClient
Imports System.Linq

Module LinqToDataSet
  Sub Main()
     Dim connectString As String = System.Configuration.ConfigurationManager.ConnectionSt

     Dim sqlSelect As String = "SELECT * FROM Department;" + "SELECT * FROM Employee;"
     Dim sqlCnn As SqlConnection = New SqlConnection(connectString)
     sqlCnn.Open()

     Dim da As New SqlDataAdapter
     da.SelectCommand = New SqlCommand(sqlSelect, sqlCnn)

     da.TableMappings.Add("Table", "Department")
     da.TableMappings.Add("Table1", "Employee")

     Dim ds As New DataSet()
     da.Fill(ds)

     Dim dr As DataRelation = ds.Relations.Add("FK_Employee_Department", ds.Tables("Depar

     Dim department As DataTable = ds.Tables("Department")
     Dim employee As DataTable = ds.Tables("Employee")

     Dim query = From d In department.AsEnumerable()
                 Join e In employee.AsEnumerable() On d.Field(Of Integer)("DepartmentId")
                 e.Field(Of Integer)("DepartmentId")
                 Select New Person With{ _
                        .EmployeeId = e.Field(Of Integer)("EmployeeId"),
                        .EmployeeName = e.Field(Of String)("Name"),
                        .DepartmentId = d.Field(Of Integer)("DepartmentId"),
                        .DepartmentName = d.Field(Of String)("Name")
                                       }

     For Each e In query
        Console.WriteLine("Employee Id = {0} , Name = {1} , Department Name = {2}", e.Emp
     Next

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub

  Class Person
     Public Property EmployeeId As Integer
     Public Property EmployeeName As String
     Public Property DepartmentId As Integer
     Public Property DepartmentName As String
  End Class
End Module
```

当C#或VB的上述代码被编译和执行时，它产生了以下结果：

```
Employee Id = 1, Name = William, Department Name = Account
Employee Id = 2, Name = Benjamin, Department Name = Account
Employee Id = 3, Name = Miley, Department Name = Sales

Press any key to continue.
```

使用LINQ到数据集查询数据集

在开始查询使用LINQ到数据集数据集，这是至关重要的数据加载到数据集，这是通过或者使用DataAdapter类或LINQ到SQL完成的。使用LINQ到数据集查询的提法和通过使用LINQ与其他LINQ使数据源制定查询是非常相似的。

单表查询

在下面的单表查询，所有的在线订单从SalesOrderHeaderTtable收集，然后命令ID，订购日期和订单号显示为输出。

C

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LinqToDataset
{
  class SingleTable
  {
     static void Main(string[] args)
     {
        string connectString = System.Configuration.ConfigurationManager.ConnectionString

        string sqlSelect = "SELECT * FROM Department;";

        // Create the data adapter to retrieve data from the database
        SqlDataAdapter da = new SqlDataAdapter(sqlSelect, connectString);

        // Create table mappings
        da.TableMappings.Add("Table", "Department");

        // Create and fill the DataSet
        DataSet ds = new DataSet();
        da.Fill(ds);

        DataTable department = ds.Tables["Department"];

        var query = from d in department.AsEnumerable()
        select new
                 {
                    DepartmentId = d.Field<int>("DepartmentId"),
                    DepartmentName = d.Field<string>("Name")
                 };

        foreach (var q in query)
        {
           Console.WriteLine("Department Id = {0} , Name = {1}",
                             q.DepartmentId, q.DepartmentName);
        }

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Imports System.Data.SqlClient
Imports System.Linq

Module LinqToDataSet
  Sub Main()
     Dim connectString As String = System.Configuration.ConfigurationManager.ConnectionSt

     Dim sqlSelect As String = "SELECT * FROM Department;"
     Dim sqlCnn As SqlConnection = New SqlConnection(connectString)
     sqlCnn.Open()

     Dim da As New SqlDataAdapter
     da.SelectCommand = New SqlCommand(sqlSelect, sqlCnn)

     da.TableMappings.Add("Table", "Department")
     Dim ds As New DataSet()
     da.Fill(ds)

     Dim department As DataTable = ds.Tables("Department")

     Dim query = From d In department.AsEnumerable()
     Select New DepartmentDetail With {
                                         .DepartmentId = d.Field(Of Integer)("DepartmentId
                                         .DepartmentName = d.Field(Of String)("Name")
                                       }

     For Each e In query
        Console.WriteLine("Department Id = {0} , Name = {1}", e.DepartmentId, e.Departmen
     Next

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub

  Public Class DepartmentDetail
     Public Property DepartmentId As Integer
     Public Property DepartmentName As String
  End Class
End Module
```

当C#或VB的上述代码被编译和执行时，它产生了以下结果：

```
Department Id = 1, Name = Account
Department Id = 2, Name = Sales
Department Id = 3, Name = Pre-Sales
Department Id = 4, Name = Marketing

Press any key to continue.
```

LINQ XML - LinQ教程

LINQ到XML提供易于访问所有的LINQ功能类似于标准查询操作，编程接口等。 集成在.NET框架的LINQ to XML也使得物尽其用的.NET框架的功能像调试，编译时检查，强类型。

LINQ 到 XML介绍

虽然使用LINQ 到 XML，加载XML文档到内存容易，更容易被查询和修改文档。另外，也可以保存在现有的存储器磁盘的XML文档，并对其进行序列化。它省去了一个开发人员学习XML查询语言这是有点复杂的。

LINQ到XML有它在System.Xml.Linq的命名。这是过去19个必要的类使用XML。这些类如以下所示。

- XAttribute
- XCData
- XComment
- XContainer
- XDeclaration
- XDocument
- XDocumentType
- XElement
- XName
- XNamespace
- XNode
- XNodeDocumentOrderComparer
- XNodeEqualityComparer
- XObject
- XObjectChange
- XObjectChangeEventArgs
- XObjectEventHandler
- XProcessingInstruction
- XText

使用LINQ读取XML文件

C

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace LINQtoXML
{
  class ExampleOfXML
  {
     static void Main(string[] args)
     {
        string myXML = @"<Departments>
                       <Department>Account</Department>
                       <Department>Sales</Department>
                       <Department>Pre-Sales</Department>
                       <Department>Marketing</Department>
                       </Departments>";

        XDocument xdoc = new XDocument();
        xdoc = XDocument.Parse(myXML);

        var result = xdoc.Element("Departments").Descendants();

        foreach (XElement item in result)
        {
           Console.WriteLine("Department Name - " + item.Value);
        }

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Xml.Linq

Module Module1
  Sub Main(ByVal args As String())
     Dim myXML As String = "<Departments>" & vbCr & vbLf &
                           "<Department>Account</Department>" & vbCr & vbLf &
                           "<Department>Sales</Department>" & vbCr & vbLf &
                           "<Department>Pre-Sales</Department>" & vbCr & vbLf &
                           "<Department>Marketing</Department>" & vbCr & vbLf & "</Depart

     Dim xdoc As New XDocument()
     xdoc = XDocument.Parse(myXML)

     Dim result = xdoc.Element("Departments").Descendants()

     For Each item As XElement In result
        Console.WriteLine("Department Name - " + item.Value)
     Next

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub
End Module
```

当C＃或VB的上述代码被编译和执行时，它产生了以下结果：

```
Department Name - Account
Department Name - Sales
Department Name - Pre-Sales
Department Name - Marketing

Press any key to continue.
```

添加新节点

C

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace LINQtoXML
{
  class ExampleOfXML
  {
     static void Main(string[] args)
     {
        string myXML = @"<Departments>
                       <Department>Account</Department>
                       <Department>Sales</Department>
                       <Department>Pre-Sales</Department>
                       <Department>Marketing</Department>
                       </Departments>";

        XDocument xdoc = new XDocument();
        xdoc = XDocument.Parse(myXML);

        //Add new Element
        xdoc.Element("Departments").Add(new XElement("Department", "Finance"));

        //Add new Element at First
        xdoc.Element("Departments").AddFirst(new XElement("Department", "Support"));

        var result = xdoc.Element("Departments").Descendants();

        foreach (XElement item in result)
        {
           Console.WriteLine("Department Name - " + item.Value);
        }

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Xml.Linq

Module Module1
  Sub Main(ByVal args As String())
     Dim myXML As String = "<Departments>" & vbCr & vbLf &
                           "<Department>Account</Department>" & vbCr & vbLf &
                           "<Department>Sales</Department>" & vbCr & vbLf &
                           "<Department>Pre-Sales</Department>" & vbCr & vbLf &
                           "<Department>Marketing</Department>" & vbCr & vbLf &
                           "</Departments>"

     Dim xdoc As New XDocument()
     xdoc = XDocument.Parse(myXML)

     xdoc.Element("Departments").Add(New XElement("Department", "Finance"))

     xdoc.Element("Departments").AddFirst(New XElement("Department", "Support"))

     Dim result = xdoc.Element("Departments").Descendants()

     For Each item As XElement In result
        Console.WriteLine("Department Name - " + item.Value)
     Next

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub
End Module
```

当C＃或VB的上述代码被编译和执行时，它产生了以下结果：

```
Department Name - Support
Department Name - Account
Department Name - Sales
Department Name - Pre-Sales
Department Name - Marketing
Department Name - Finance

Press any key to continue.
```

删除特定节点

C

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace LINQtoXML
{
  class ExampleOfXML
  {
     static void Main(string[] args)
     {
        string myXML = @"<Departments>
                       <Department>Support</Department>
                       <Department>Account</Department>
                       <Department>Sales</Department>
                       <Department>Pre-Sales</Department>
                       <Department>Marketing</Department>
                       <Department>Finance</Department>
                       </Departments>";

        XDocument xdoc = new XDocument();
        xdoc = XDocument.Parse(myXML);

        //Remove Sales Department
        xdoc.Descendants().Where(s =>s.Value == "Sales").Remove();

        var result = xdoc.Element("Departments").Descendants();

        foreach (XElement item in result)
        {
           Console.WriteLine("Department Name - " + item.Value);
        }

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

VB

```
Imports System.Collections.Generic
Imports System.Linq
Imports System.Xml.Linq

Module Module1
  Sub Main(args As String())
     Dim myXML As String = "<Departments>" & vbCr & vbLf &
                            "<Department>Support</Department>" & vbCr & vbLf &
                            "<Department>Account</Department>" & vbCr & vbLf &
                            "<Department>Sales</Department>" & vbCr & vbLf &
                            "<Department>Pre-Sales</Department>" & vbCr & vbLf &
                            "<Department>Marketing</Department>" & vbCr & vbLf &
                            "<Department>Finance</Department>" & vbCr & vbLf & "</Departm

     Dim xdoc As New XDocument()
     xdoc = XDocument.Parse(myXML)

     xdoc.Descendants().Where(Function(s) s.Value = "Sales").Remove()

     Dim result = xdoc.Element("Departments").Descendants()

     For Each item As XElement In result
        Console.WriteLine("Department Name - " + item.Value)
     Next

     Console.WriteLine(vbLf & "Press any key to continue.")
     Console.ReadKey()
  End Sub
End Module
```

当C＃或VB的上述代码被编译和执行时，它产生了以下结果：

```
Department Name - Support
Department Name - Account
Department Name - Pre-Sales
Department Name - Marketing
Department Name - Finance

Press any key to continue.
```

LINQ实体 - LinQ教程

作为ADO.NET实体框架的一部分，LINQ到实体比LINQ到SQL更灵活，但由于其复杂性和缺乏关键功能没有太大的普及。但是，它没有LINQ的限制SQL允许数据查询，只能在SQL服务器数据库LINQ到实体中查询大量数据，如Oracle，MySQL等资料提供方便数据查询

此外，它已经得到了在这个意义上，用户可以使用一个数据源控件执行通过LINQ到实体查询，结果没有任何需要额外的编码有利于结合从ASP.Net支持。

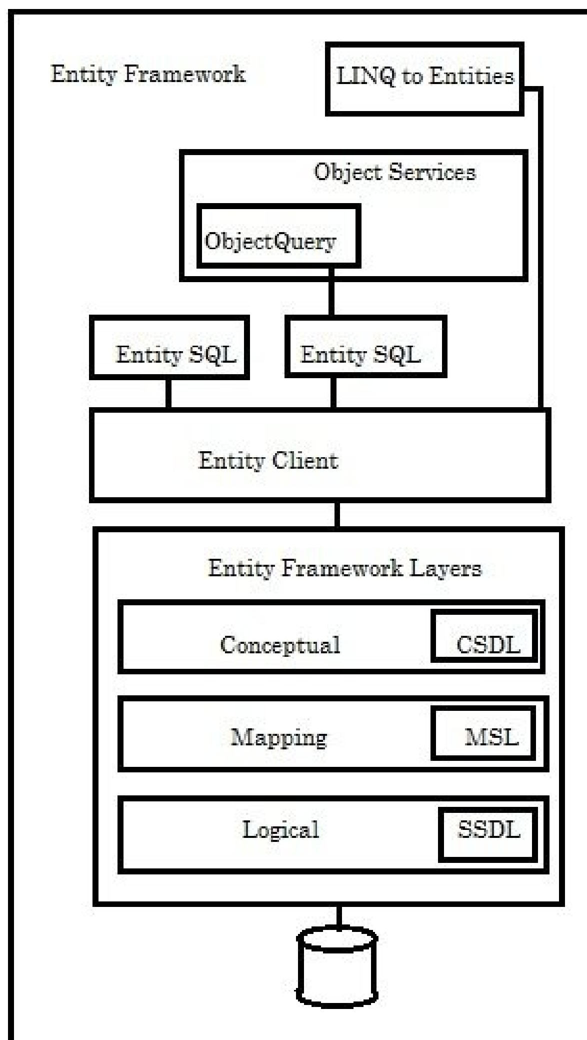
LINQ到实体具有这些优点成为了标准机制，LINQ对数据库的使用。另外，也可以与LINQ实体来改变查询的数据的信息和容易批量更新。最有趣的是有关LINQ到实体具有像SQL相同的语法，甚至有同组标准查询运算符，如加入，选择，排序等等。

LINQ 到Entities查询的创建和执行过程

- 建设一个ObjectQuery的实例ObjectContext（实体连接）
- 通过使用新建成的情况下构成C#或Visual Basic（VB）查询
- 转换LINQ的标准查询操作符，以及LINQ表达式到命令树
- 执行直接传递到遇到客户端任何异常查询
- 返回到客户机的所有查询结果

ObjectContext是这里的主类，使与实体数据模型或换句话说相互作用充当连接LINQ到数据库的桥. 命令树是这里与实体框架兼容的查询表示。 实体框架，另一方面实际上是对象关系映射器一般简称ORM，由做业务对象的产生，以及实体按照数据库表，便于各种基本操作，如创建，更新，删除和读取。

下面是一个图表实体框架到更好地了解这一概念的。

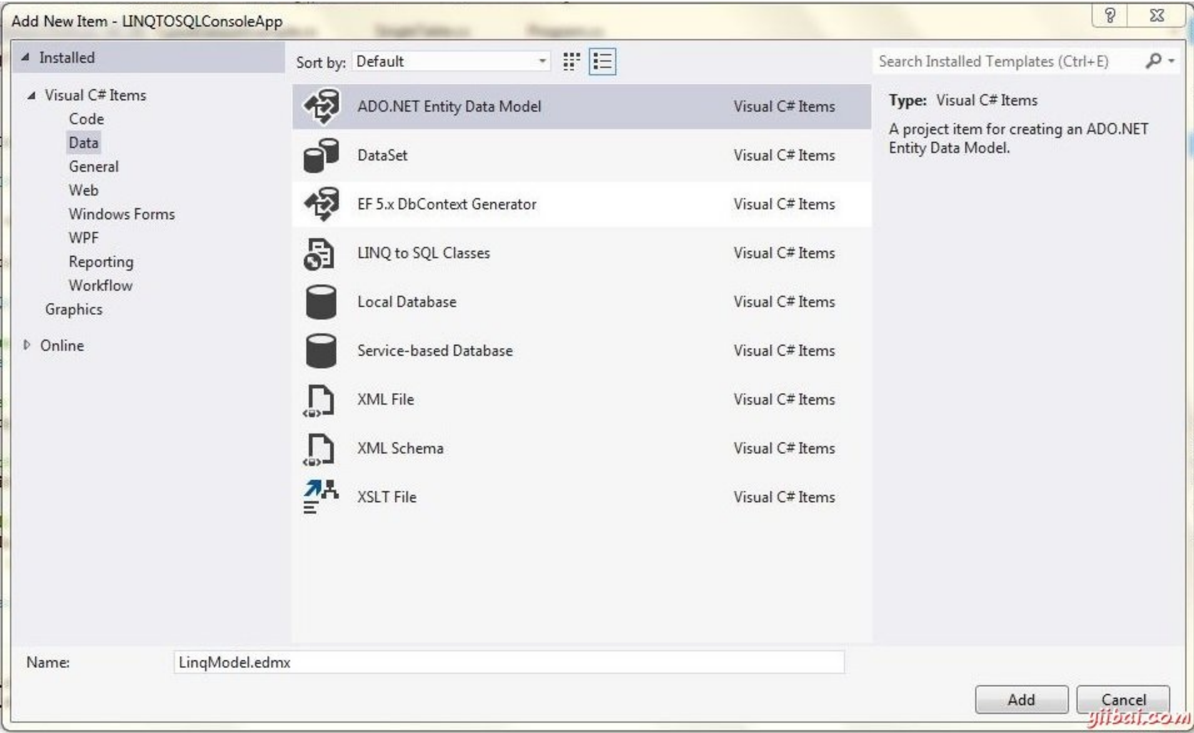


使用LINQ实体模型添加，更新和删除示例

首先添加实体模型按以下步骤。

- 步骤 1: 右键单击项目，然后单击添加新项目将打开的窗口中按如下。选择ADO.NET实体数据模型，并指定名称，然后单击添加。

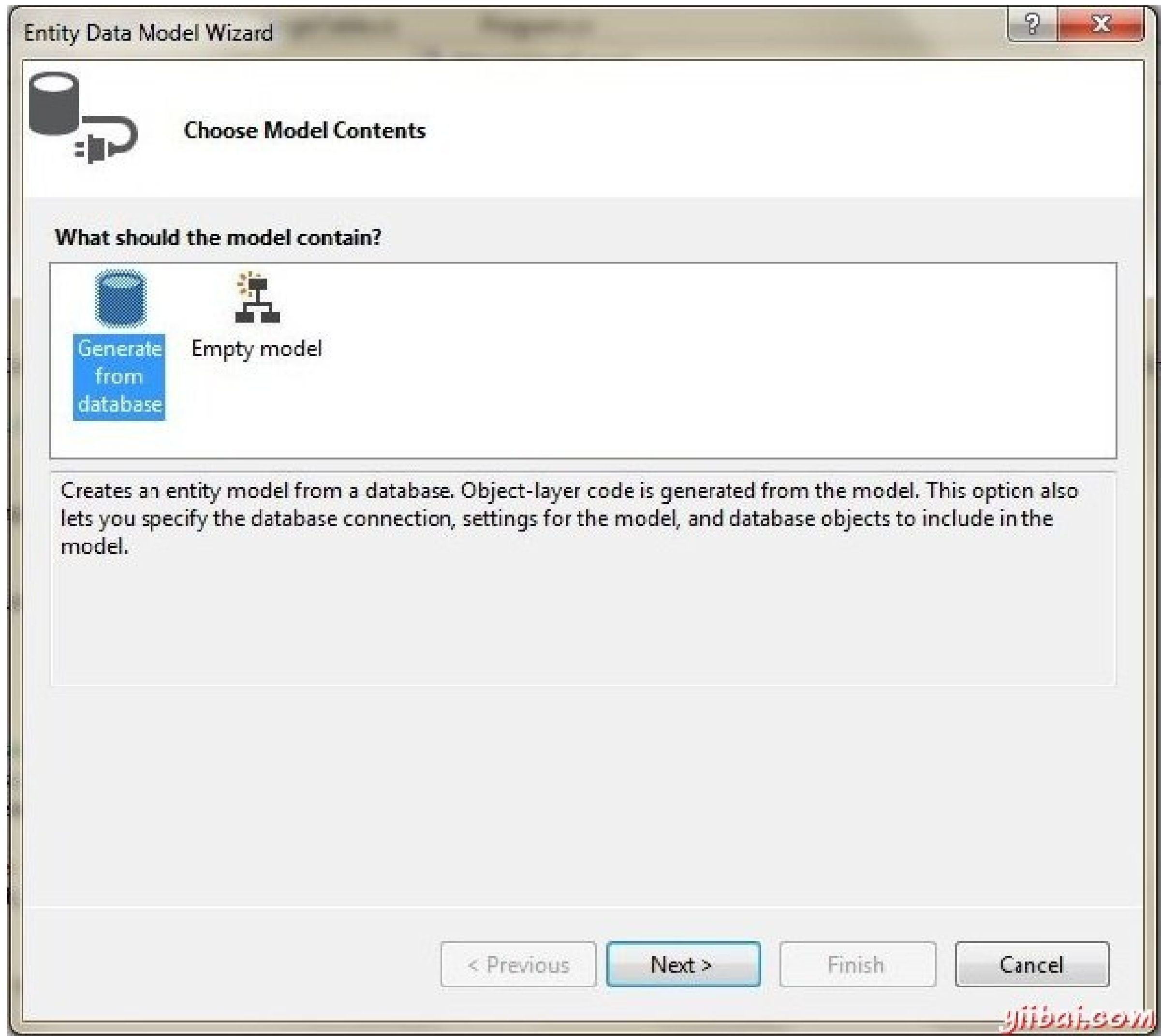
<center style="box-sizing: border-box;">



</center>

- 步骤 2: 选择从数据库生成。

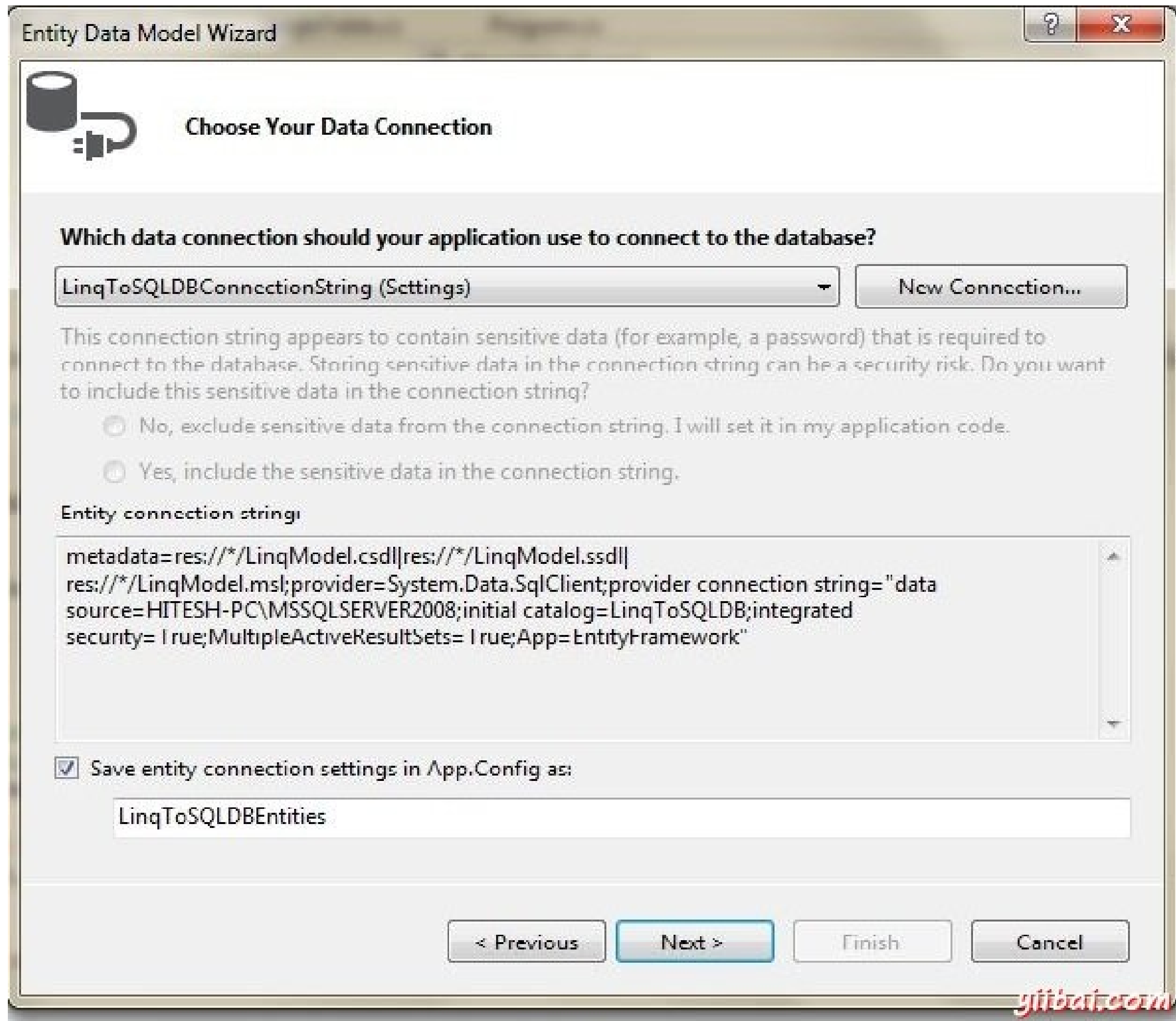
<center style="box-sizing: border-box;">



</center>

- 步骤 3: 选择数据库连接。

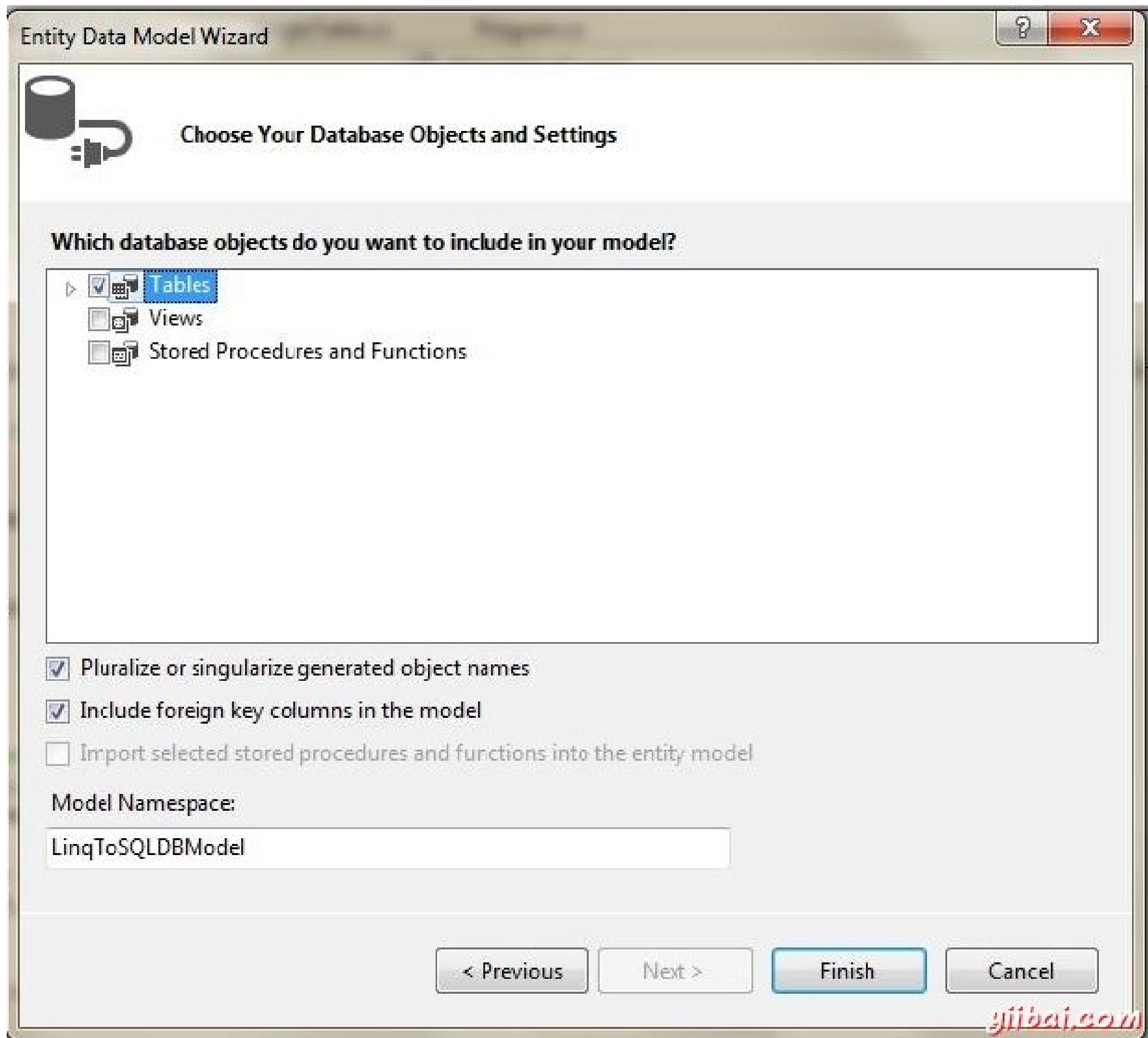
<center style="box-sizing: border-box;">



</center>

- 步骤 4: 选择所有表

<center style="box-sizing: border-box;">



</center>

现在写下面的代码。

```
using DataAccess;
using System;
using System.Linq;

namespace LINQTOSQLConsoleApp
{
  public class LinqToEntityModel
  {
     static void Main(string[] args)
     {
        using (LinqToSQLDBEntities context = new LinqToSQLDBEntities())
        {
           //Get the List of Departments from Database
           var departmentList = from d in context.Departments
           select d;

           foreach (var dept in departmentList)
           {
              Console.WriteLine("Department Id = {0} , Department Name = {1}",
                                  dept.DepartmentId, dept.Name);
           }

           //Add new Department
           DataAccess.Department department = new DataAccess.Department();
           department.Name = "Support";

           context.Departments.Add(department);
           context.SaveChanges();

           Console.WriteLine("Department Name = Support is inserted in Database");

           //Update existing Department
           DataAccess.Department updateDepartment = context.Departments.FirstOrDefault(d
           updateDepartment.Name = "Account updated";
           context.SaveChanges();

           Console.WriteLine("Department Name = Account is updated in Database");

           //Delete existing Department
           DataAccess.Department deleteDepartment = context.Departments.FirstOrDefault(d
           context.Departments.Remove(deleteDepartment);
           context.SaveChanges();

           Console.WriteLine("Department Name = Pre-Sales is deleted in Database");

           //Get the Updated List of Departments from Database
           departmentList = from d in context.Departments
           select d;

           foreach (var dept in departmentList)
           {
              Console.WriteLine("Department Id = {0} , Department Name = {1}",
                                  dept.DepartmentId, dept.Name);
           }
        }

        Console.WriteLine("\nPress any key to continue.");
        Console.ReadKey();
     }
  }
}
```

让我们编译和运行上面的程序，这将产生以下结果：

file:///F:/Data/Projects/LINQTOSQLConsoleApp/LINQTOSQLConsoleApp/bin/Debug/LINQTOSQLC...

```
Department Id = 1 , Department Name = Account
Department Id = 2 , Department Name = Sales
Department Id = 3 , Department Name = Pre-Sales
Department Id = 4 , Department Name = Marketing
Department Name = Support is inserted in Database
Department Name = Account is updated in Database
Department Name = Pre-Sales is deleted in Database
Department Id = 1 , Department Name = Account updated
Department Id = 2 , Department Name = Sales
Department Id = 4 , Department Name = Marketing
Department Id = 5 , Department Name = Support

Press any key to continue.
```

LINQ Lambda表达式 - LinQ教程

术语“Lambda表达式'是来自于”拉姆达“演算这又是施加用于定义功能的数学符号及其名称。lambda表达式作为LINQ式的可执行部分转换的方式在运行时的逻辑，因此它可以传递到数据源方便。但是，lambda表达式不仅仅局限于查找应用LINQ而已。

这些表达式由下面的语法表示：

(输入参数) => 表达式或语句块

下面是lambda表达式的一个例子

y => y * y

上述表达式指定一个参数y和y的值的平方。然而，这是不可能的到这种形式来执行一个lambda表达式。在C#中的lambda表达式的例子如下所示。

C

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace lambdaexample
{
  class Program
  {
     delegate int del(int i);
     static void Main(string[] args)
     {
        del myDelegate = y => y * y;
        int j = myDelegate(5);
        Console.WriteLine(j);
        Console.ReadLine();
     }
  }
}
```

VB

```
Module Module1
  Private Delegate Function del(ByVal i As Integer) As Integer
  Sub Main(ByVal args As String())
     Dim myDelegate As del = Function(y) y * y
     Dim j As Integer = myDelegate(5)
     Console.WriteLine(j)
     Console.ReadLine()
  End Sub
End Module
```

当C#或VB的上述代码被编译和执行时，它产生了以下结果：

```
25
```

表达Lambda

如在lambda表达式的上面所示的语法表达是在右手侧，这些也被称为lambda表达式。

异步Lambdas

通过使用异步关键字结合异步处理创建lambda表达式被称为异步lambda表达式。下面是异步lambda的一个例子。

```
Func<Task<string>> getWordAsync = async() => "hello";
```

Lambda的标准查询操作

查询运算符内的lambda表达通过在需要时相同的评估计算，并不断地作用于各输入序列中的元素，而不是整个序列。 开发人员允许Lambda表达式到自己的逻辑转换成标准查询操作.在下面的例子中，开发人员使用的"Where"运算符，通过利用lambda表达式的回收，从给出的列表中奇数值。

C

```
//Get the average of the odd Fibonacci numbers in the series...
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace lambdaexample
{
  class Program
  {
     static void Main(string[] args)
     {
        int[] fibNum = { 1, 1, 2, 3, 5, 8, 13, 21, 34 };
        double averageValue = fibNum.Where(num => num % 2 == 1).Average();
        Console.WriteLine(averageValue);
        Console.ReadLine();
     }
  }
}
```

VB

```
Module Module1
  Sub Main()
     Dim fibNum As Integer() = {1, 1, 2, 3, 5, 8, 13, 21, 34}
     Dim averageValue As Double = fibNum.Where(Function(num) num Mod 2 = 1).Average()
     Console.WriteLine(averageValue)
     Console.ReadLine()
  End Sub
End Module
```

让我们编译和运行上面的程序，这将产生以下结果：

```
7.33333333333333
```

lambda类型推断

在C#中，类型推断方便地用于各种情况，而且也没有明确指定的类型。但是如果一个lambda表达式，类型推断将工作必须满足在已指定每种类型为编译器。让我们考虑下面的例子。

```
delegate int Transformer (int i);
```

这里，编译器采用的类型推理时的事实，x是一个整数，这是通过检查所述变压器的参数类型进行绘制。

Lambda表达式变量的作用域

有而这样的lambda表达式内发起变量的lambda表达式，使用变量的作用域并不意味着是可见的外部方法有一些规则。还有一个规则，一个捕获变量不是被垃圾回收，除非委托引用相同变得符合垃圾收集的行为。此外，还有禁止lambda表达式中的return语句导致返回封闭方法的规则。

这里有一个例子，以证明lambda表达式变量的作用域。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace lambdaexample
{
  class Program
  {
     delegate bool D();
     delegate bool D2(int i);

     class Test
     {
        D del;
        D2 del2;
        public void TestMethod(int input)
        {
           int j = 0;
           // Initialize the delegates with lambda expressions.
           // Note access to 2 outer variables.
           // del will be invoked within this method.
           del = () => { j = 10; return j > input; };

           // del2 will be invoked after TestMethod goes out of scope.
           del2 = (x) => { return x == j; };

           // Demonstrate value of j:
           // The delegate has not been invoked yet.
           Console.WriteLine("j = {0}", j);        // Invoke the delegate.
           bool boolResult = del();

           Console.WriteLine("j = {0}. b = {1}", j, boolResult);
        }

        static void Main()
        {
           Test test = new Test();
           test.TestMethod(5);

           // Prove that del2 still has a copy of
           // local variable j from TestMethod.
           bool result = test.del2(10);

           Console.WriteLine(result);

           Console.ReadKey();
        }
     }
  }
}
```

让我们编译和运行上面的程序，这将产生以下结果：

```
j = 0
j = 10\. b = True
True
```

表达式树

Lambda表达式中使用表达式树结构广泛。表达式树放弃代码中的数据结构类似于树，其中每个节点本身是一样的方法调用的表达，或者可以是一个二进制运算如x<y，下面是lambda表达式的使用用于构造一个表达式树的一个例子。

LAMBDA语句

还有lambda表达式由两个或三个语句，但不仅在构造表达式树中。 return语句必须写在lambda语句。

lambda声明的语法

```
(params) => {statements}
```

lambda的声明示例

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Linq.Expressions;

namespace lambdaexample
{
  class Program
  {
     static void Main(string[] args)
     {
        int[] source = new[] { 3, 8, 4, 6, 1, 7, 9, 2, 4, 8 };

        foreach (int i in source.Where(
                 x =>
                 {
                    if (x <= 3)
                       return true;
                    else if (x >= 7)
                       return true;
                    return false;
                 }
                ))
        Console.WriteLine(i);
        Console.ReadLine();
     }
  }
}
```

让我们编译和运行上面的程序，这将产生以下结果：

```
3
8
1
7
9
2
8
```

lambda表达式被用作基于方法的LINQ查询参数，并决不允许有一个地方对操作的左侧像是或者就像匿名方法。 虽然，Lambda表达式何其相似匿名方法，这些根本不是限制被用来作为唯一表示。

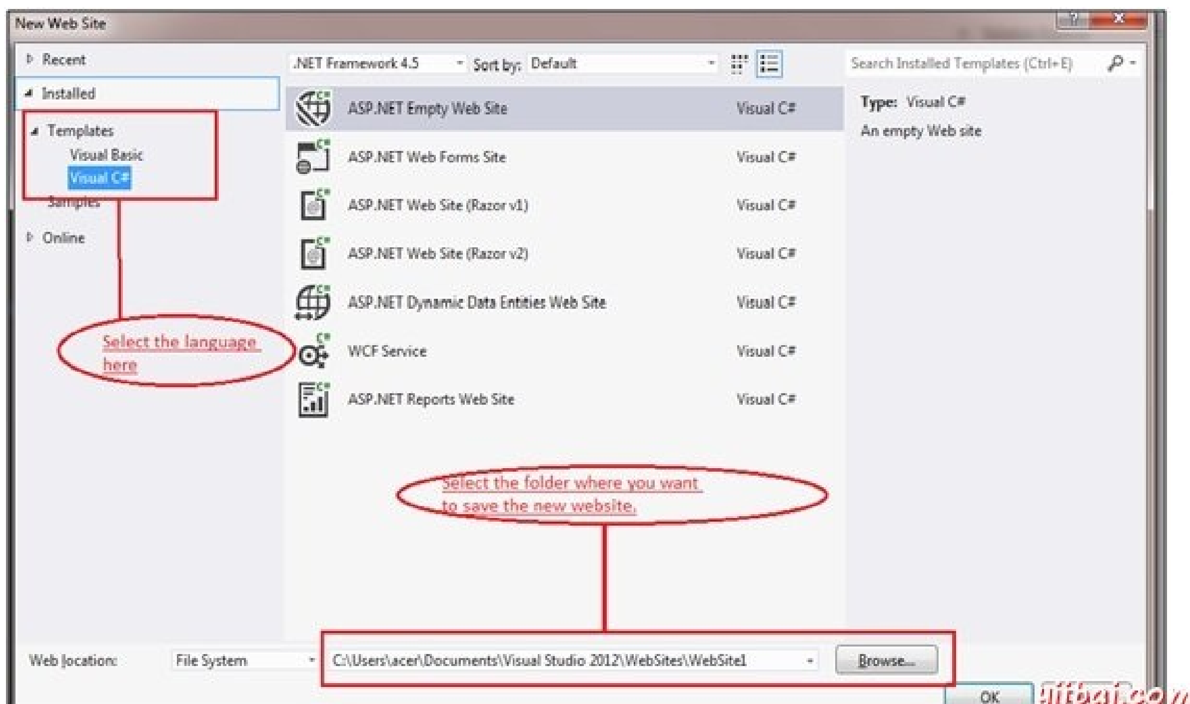
使用lambda表达式要点要记住

- lambda表达式可以返回一个值，并可以带有参数。
- 参数可以用不同的方式使用lambda表达式无数的定义。
- 如果在一个lambda表达式单独的语句，就没有必要花括号，而如果有多个语句，大括号以及返回值都是写必不可少的。
- 随着lambda表达式，可以通过被称为闭合的特征来访问变量lambda表达式块的存在之外。利用闭合的应谨慎，以避免任何问题。
- 这是不执行任何lambda表达式中的任何不安全的代码。
- lambda表达式并不意味着使用操作符的左侧。

LINQ ASP.Net - LinQ教程

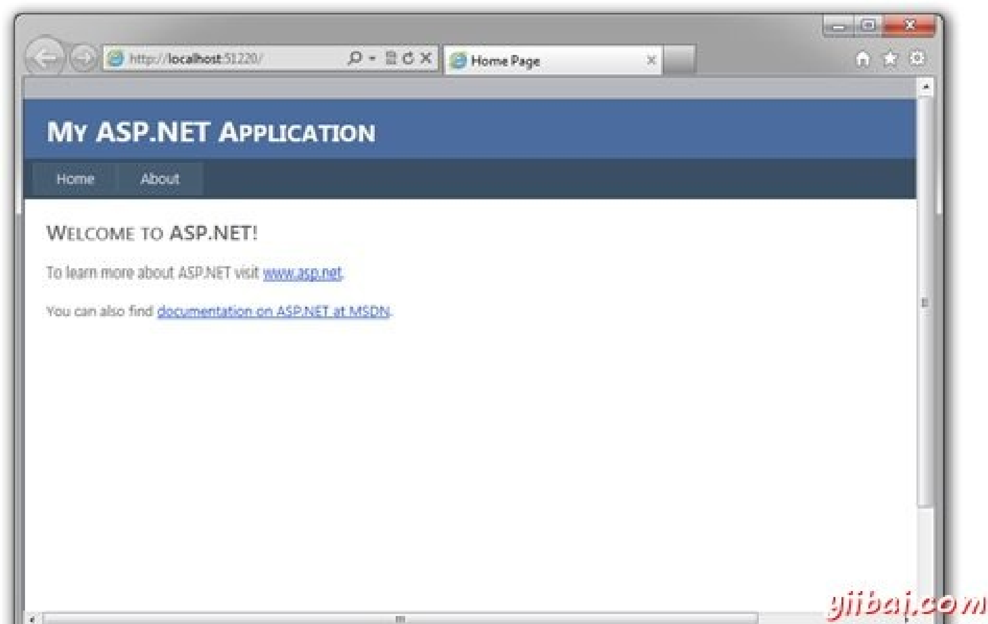
作为一组.NET框架的扩展，LINQ是首选的机制，通过ASP.NET开发人员的数据访问。ASP.NET3.5有一个内置的工具LINQDataSource控件，使LINQ容易使用ASP.NET。ASP.NET使用上述控制作为数据源。现实生活中的项目大多包括网站或Windows应用程序等到更好地了解LINQ与ASP.NET的概念，让我们开始创建一个ASP.NET网站，使用LINQ功能。

对于这一点，必须在系统上得到安装了Visual Studio和.NET框架。一旦你已经打开的Visual Studio，转到 File -> New -> Website。一个弹出窗口将在如下图所示打开。



现在根据在左手侧上的模板，将有两个语言选项创建的网站。选择Visual C#和ASP.NET选择空Web站点。

选择您要保存新的网站系统上的文件夹中。然后按确定，并很快解决方案资源管理器中出现一个包含所有网页文件在屏幕上。右键单击Default.aspx的在解决方案资源管理器，并选择查看在浏览器中查看默认ASP.NET网站在浏览器中。很快新的ASP.NET网站将得到像下图的网页浏览器中打开。



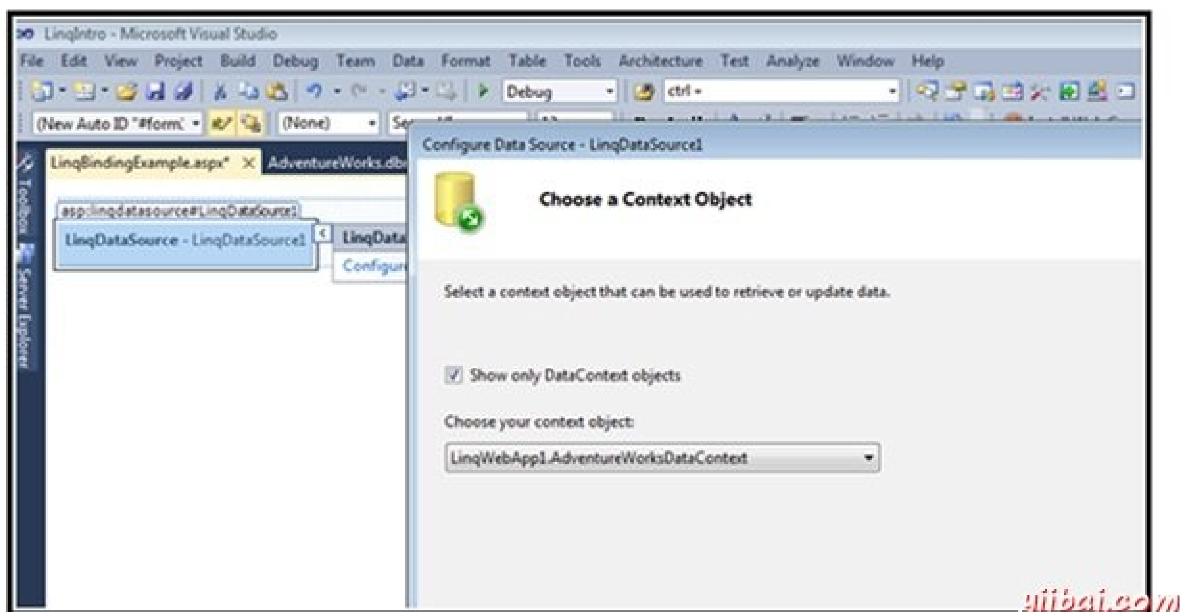
.aspx其实在ASP.NET网站上使用的主要文件扩展名。Visual Studio中默认创建的所有的基本的网站像主页和公司简介页面，您可以方便地将内容放在页面。产生用于该网站的代码自动注册，也可以查看。

LinqDataSource控件

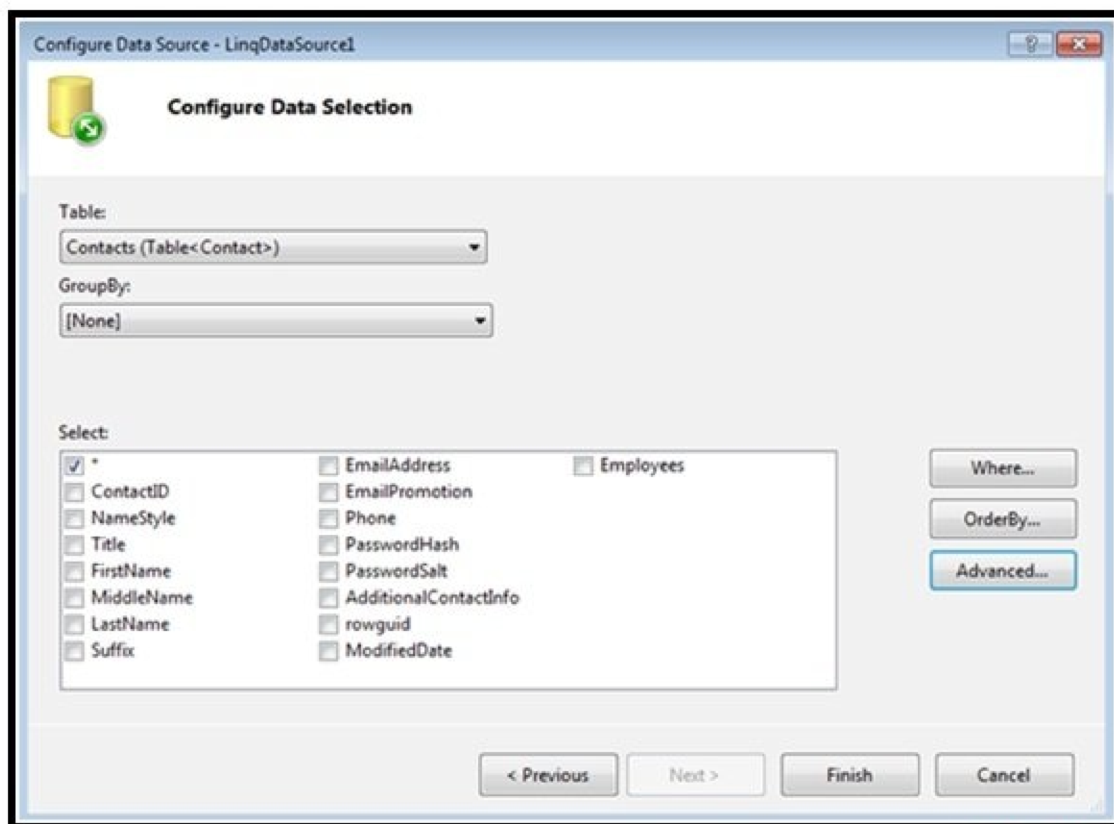
这是可能是更新，插入和在ASP.NET网站，LINQDataSource控件的帮助页面删除数据。其实完全没有必要规范SQL命令作为LINQDataSource控件采用动态对这种操作创造的命令。

控制使用户能够使一个ASP.NET网页使用LINQ的便利通过在标记文本属性设置。LinqDataSource非常相似SqlDataSource控件以及ObjectDataSource，因为它可以在结合其他ASP.NET用于控制本页到数据源上。 因此，我们必须有一个数据库来解释的LinqDataSource控件调用的各种功能。

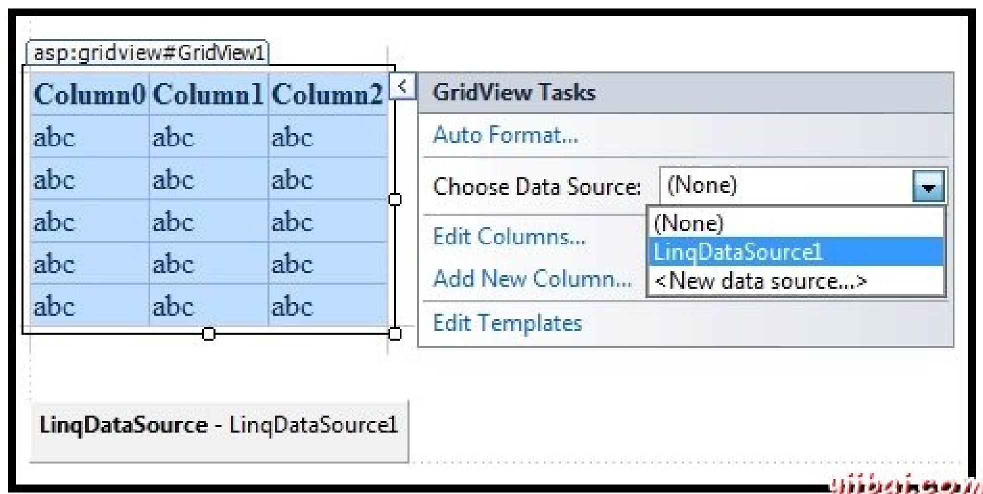
在开始控制使用ASP.NET的网页表单解释之前，有必要打开Microsoft Visual Studio工具箱拖放LINQDataSource控件到ASP.NET网站的.aspx页面就像下图。



下一步是通过选择所有列的雇员记录配置的LinqDataSource如下。



现在添加一个GridView控件到.aspx页面并进行配置，如以下图所示。GridView控件是强大的，并提供灵活的数据。很快配置控制后，它会出现在浏览器中。



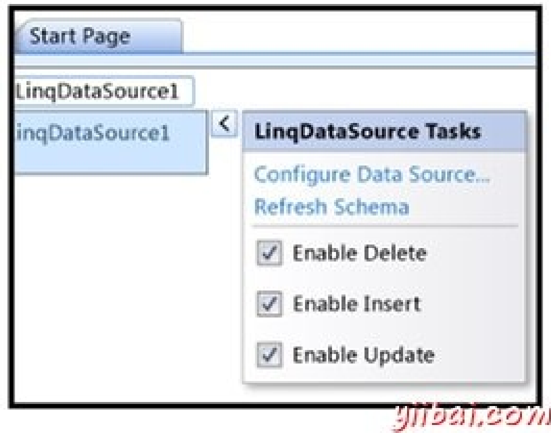
现在可以看到屏幕上为.aspx页面编码将是：

```
<!DOCTYPE html>
<html>
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
     <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
          DataKeyNames="ContactID" DataSourceID="LINQDataSource1">
        <Columns>
           <asp:BoundField DataField="ContactID" HeaderText="ContactID"
                InsertVisible="False" ReadOnly="True" SortExpression="ContactID" />
           <asp:CheckBoxField DataField="NameStyle" HeaderText="NameStyle"
                SortExpression="NameStyle" />
           <asp:BoundField DataField="Title" HeaderText="Title" SortExpression="Title" />
           <asp:BoundField DataField="FirstName" HeaderText="FirstName"
                SortExpression="FirstName" />
           <asp:BoundField DataField="MiddleName" HeaderText="MiddleName"
                SortExpression="MiddleName" />
           <asp:BoundField DataField="LastName" HeaderText="LastName"
                SortExpression="LastName" />
           <asp:BoundField DataField="Suffix" HeaderText="Suffix"
                SortExpression="Suffix" />
           <asp:BoundField DataField="EmailAddress" HeaderText="EmailAddress"
                SortExpression="EmailAddress" />
        </Columns>
     </asp:GridView>
     <br />
  </div>
  <asp:LINQDataSource ID="LINQDataSource1" runat="server"
       ContextTypeName="LINQWebApp1.AdventureWorksDataContext" EntityTypeName=""
       TableName="Contacts">
  </asp:LINQDataSource>
  </form>
</body>
</html>
```

这里应该注意的是，这是至关重要的属性ContextTypeName设置为表示数据库的类。例如，在这里它被给定为LINQWebApp1.AdventureWorksDataContext作为这个动作会使LinqDataSource和数据库之间进行所需的连接。

如何使用LINQ来INSERT，UPDATE和DELETE在ASP.NET页面的数据？

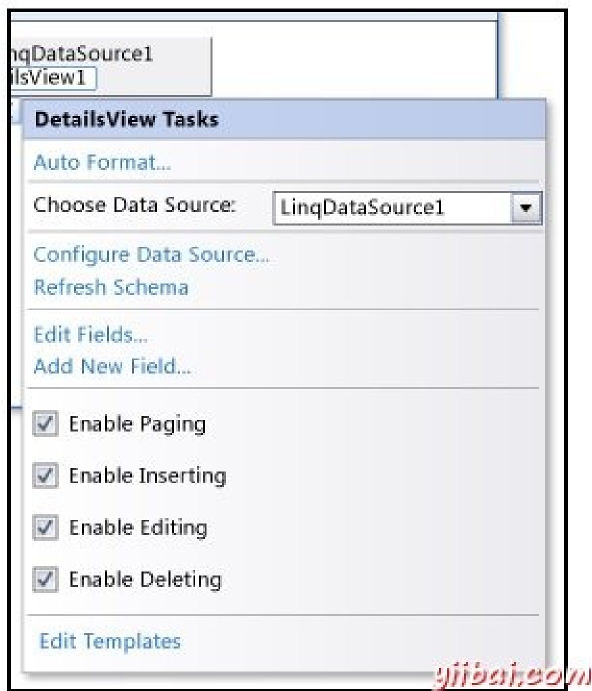
严格完成所有上述步骤后，选择从LinqDataSource控件使用LinqDataSource任务，并选择所有三个框使插入，使更新，使来自同一就像如下所示删除：



很快声明标记起来会得到显示在屏幕上如以。

```
<asp:LINQDataSource
  ContextTypeName="LINQWebApp1.AdventureWorksDataContext"
  TableName="Contacts"
  EnableUpdate="true"
  EnableInsert="true"
  EnableDelete="true"
  ID="LINQDataSource1"
  runat="server">
</asp:LINQDataSource>
```

现在，因为有多个行和列，最好是添加另一个控制你的命名为详细信息视图或主控低于网格视图控件来显示网格的选定行的只有细节的.aspx形式。选择详细信息视图控制的详细信息视图任务，选择如下图所示的复选框。



现在只需保存更改并按下Ctrl+ F5在浏览器中查看页面，现在可以删除，更新，插入的细节视图控件的记录。

VBA教程

VBA代表Visual Basic应用程序，是来自微软的事件驱动编程语言，目前主要有Microsoft Office应用程序，如MS-Excel，MS-Word和MS-Access中使用。

它可以帮助技术人员构建自定义的应用程序和解决方案，以增强这些应用程序的功能。这个设计的好处是，我们不必把Visual Basic安装我们的PC上，但安装Office将隐帮助我们达到目的。

我们可以在所有Office版本（从微软Office97至微软Office2013）直接使用，可用最新版本VBA。其中Excel的VBA是最流行的一种，并且我们可以建立在MS Excel中使用VBA非常强大的工具，包括使用线性程序。

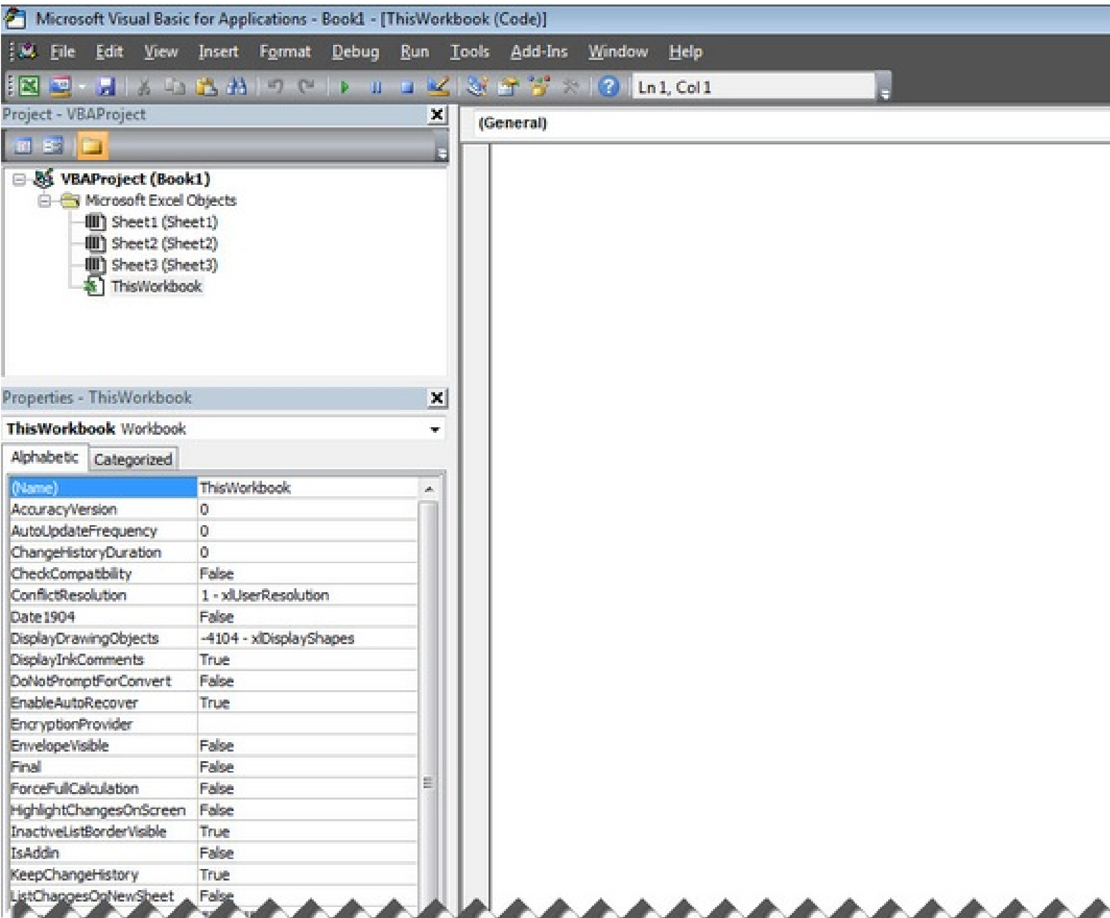
VBA的应用

为什么我们需要使用VBA在Excel中为MS-Excel本身提供了内置的功能负荷。 MS-Excel提供了唯一的基本内在功能而可能不足以执行复杂的计算。在这些情况下，VBA变成一种最明显好的解决方案。

其中一个最好的例子是非常难使用Excel内置计算贷款每月还款数，但很容易编写一个VBA这样计算。

访问VBA编辑器

在Excel窗口，按“Alt + F11”。打开VBA的窗口如下所示。



Microsoft Visual Basic for Applications - Book1 - [ThisWorkbook (Code)]
File Edit View Insert Format Debug Run Tools Add-Ins Window Help
Ln 1, Col 1
Project - VBAProject
(General)
VBAProject (Book1)
Microsoft Excel Objects
Sheet1 (Sheet1)
Sheet2 (Sheet2)
Sheet3 (Sheet3)
ThisWorkbook
Properties - ThisWorkbook
ThisWorkbook Workbook
Alphabetic
Categorized
(Name) ThisWorkbook
AccuracyVersion 0
AutoUpdateFrequency 0
ChangeHistoryDuration 0
CheckCompatibility False
ConflictResolution 1 - xlUserResolution
Date1904 False
DisplayDrawingObjects -4104 - xlDisplayShapes
DisplayInkComments True
DoNotPromptForConvert False
EnableAutoRecover True
EncryptionProvider
EnvelopeVisible False
Final False
ForceFullCalculation False
HighlightChangesOnScreen False
InactiveListBorderVisible True
IsAddin False
KeepChangeHistory True

VBA Excel宏 - VBA教程

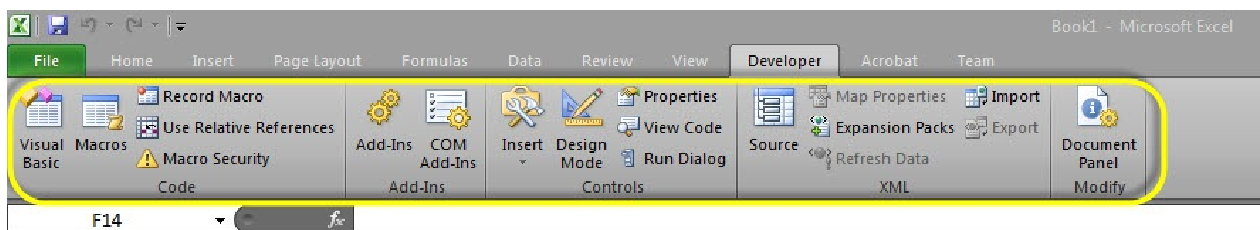
Excel VBA宏

在这一章中，让我们了解如何编写一个简单的宏。让我们一步一步来。

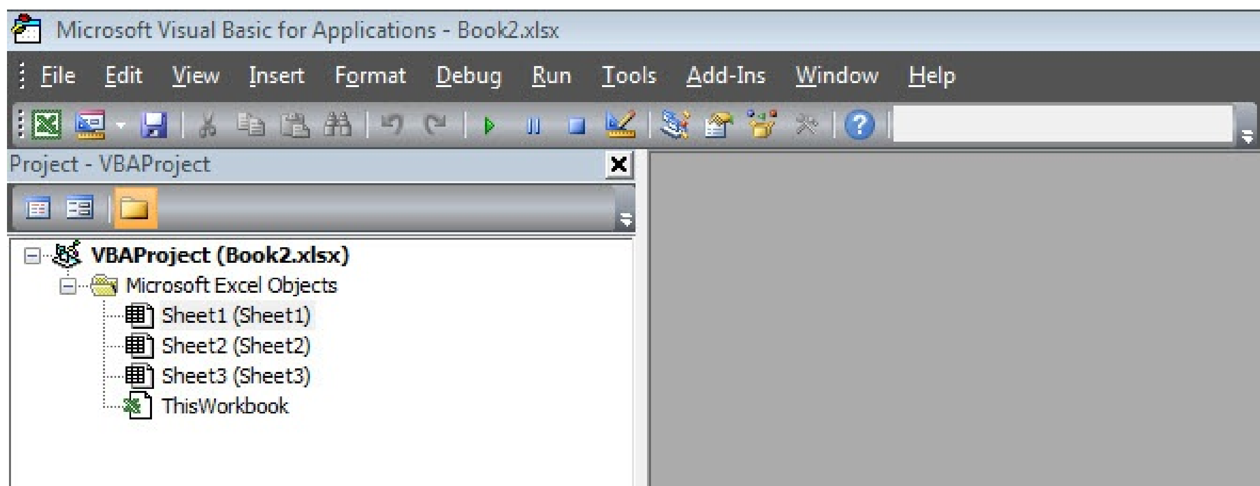
第1步：首先，让我们能够在Excel20XX'开发'菜单。做同样的，点击 File >> Option。

第2步：点击自定义功能区选项卡，并选中“Developer”，然后点击“OK”。

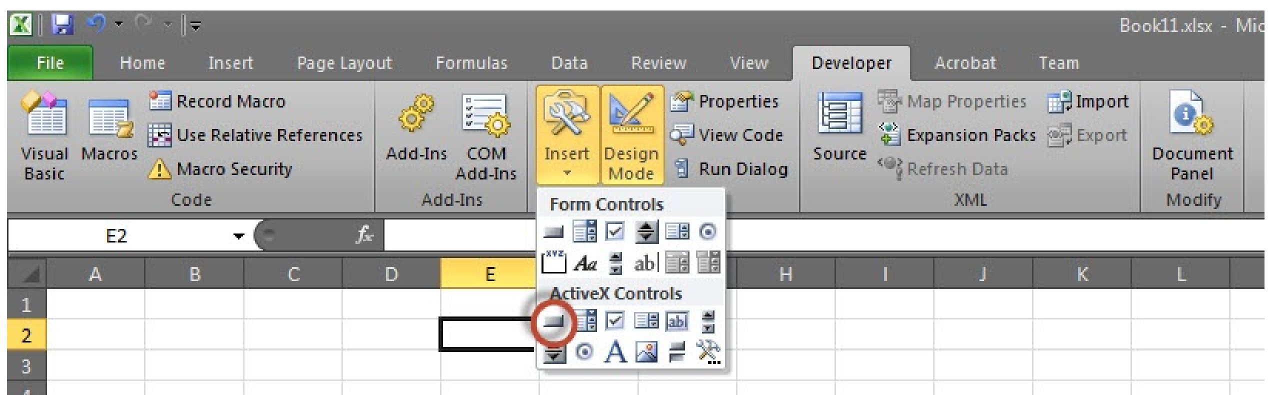
第3步：在“Developer”带状出现在菜单栏。



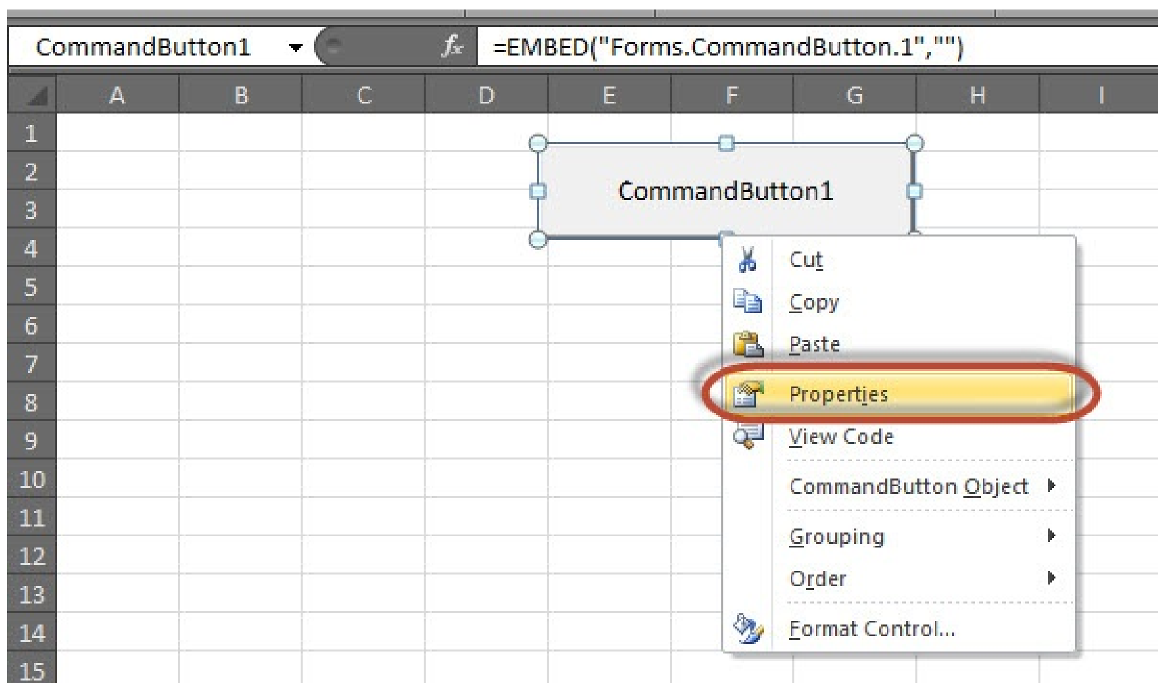
第4步：点击“Visual Basic”按钮以打开VBA编辑器。



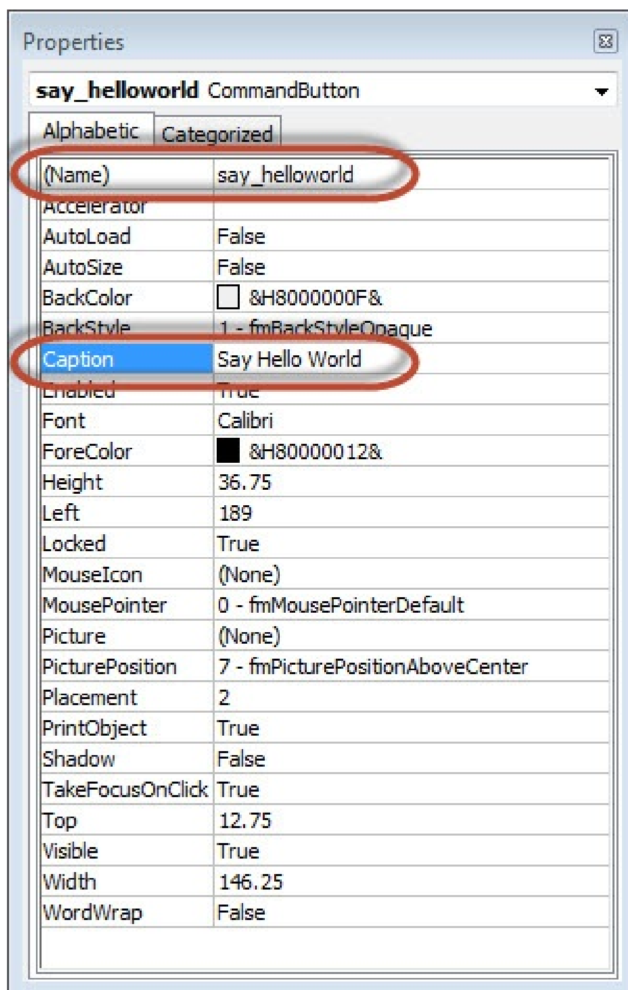
第5步：现在，让我们通过添加一个按钮，启动脚本。点击“Insert”>>选择“button”。



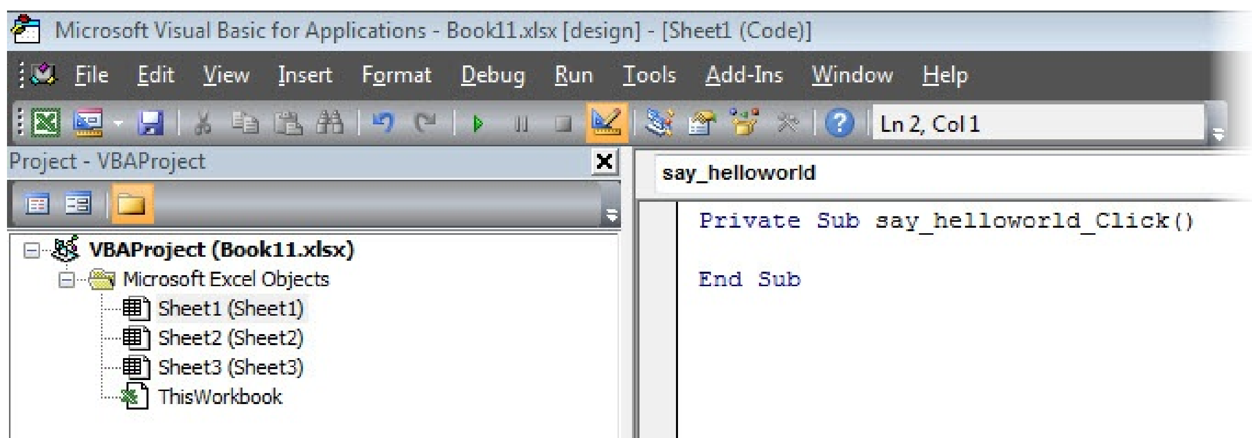
第6步：执行右键单击并选择“properties”。



第7步. 编辑名称和标题如下所示。



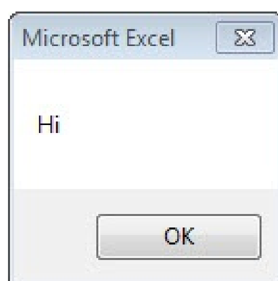
第8步：现在，双击该按钮，如下图所示的子过程的轮廓将被显示。



第9步：让我们先来简单地增加一个报文编码。

```
Private Sub say_helloworld_Click()
    MsgBox "Hi"
End Sub
```

第10步：现在，可以点击按钮执行子过程。子过程的输出如下所示。我们将会示范进一步章节使用一个简单的按钮，从步骤＃1-10已说明到。因此，彻底理解本章对以后内容的学习是很重要的。


Microsoft Excel
Hi
OK

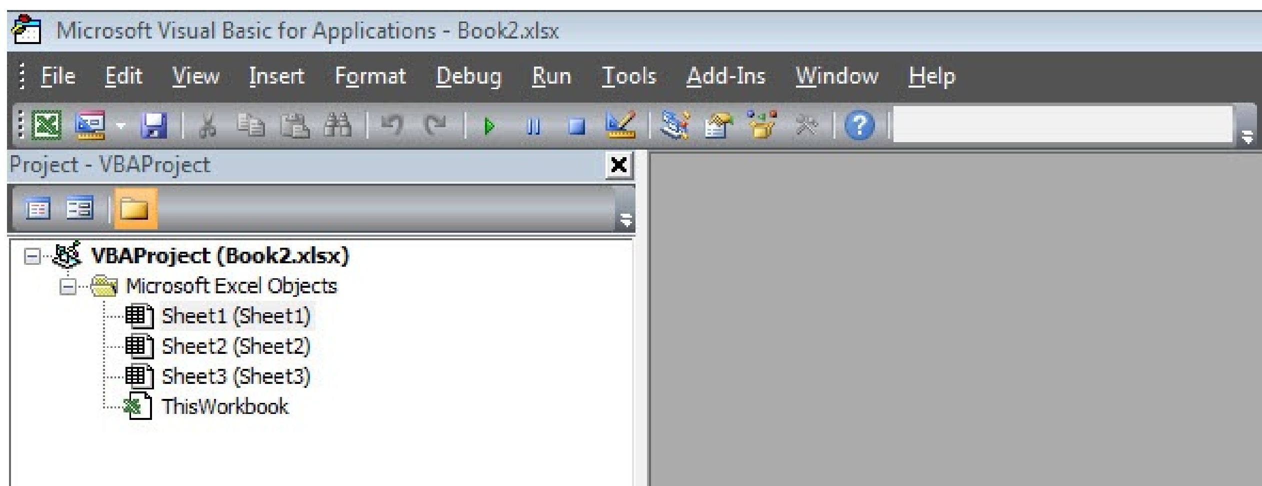
Excel VBA术语 - VBA教程

Excel VBA名词术语

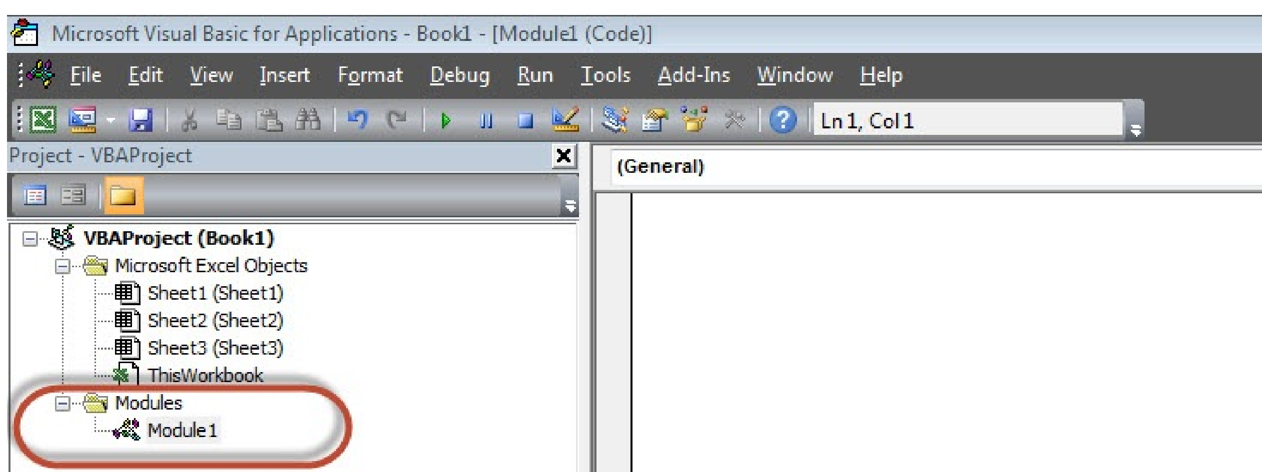
在这一章中，让我们了解常用的Excel VBA术语。这些术语将在进一步模块学习中使用，因此理解它们是非常关键的。

模块

1.模块是其中代码被写入的区域。这是一个新的工作簿，因此不会有任何模块。



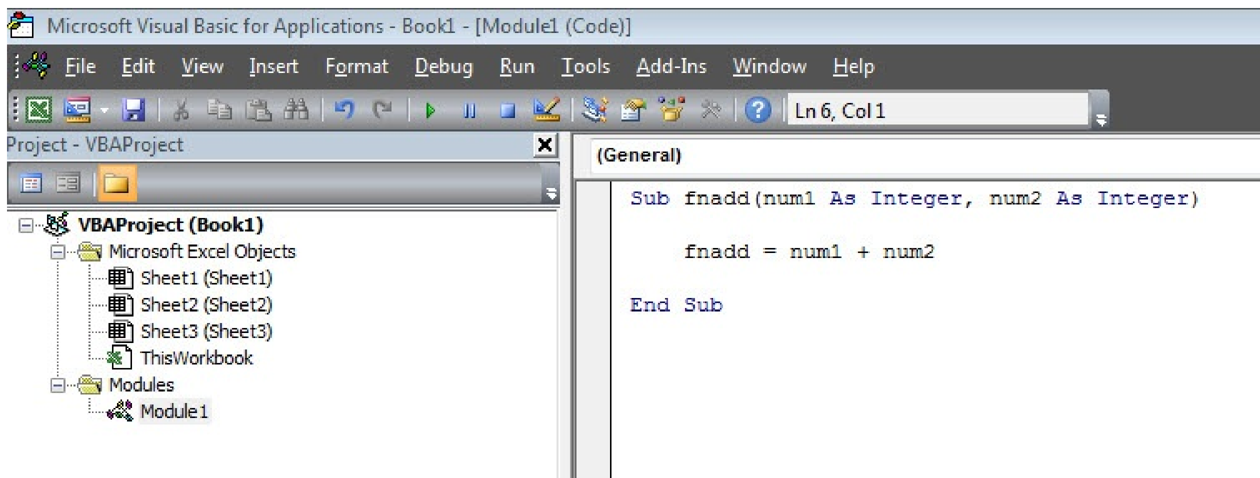
2.要插入导航模块Insert >> Module。一旦模块被插入“module1”创建。在该模块中，我们可以编写VBA代码和代码编写过程。程序/Sub过程是一系列的VBA语句指示怎么做。



过程

过程组被作为一个整体，指示Excel中如何执行特定任务执行的语句。执行的任务可以非常简单或非常复杂的，这是一个很好的做法，把程序分成较小的部分。

两种主要类型的过程分别是：Sub和Function。



函数

函数是一组可重用的代码，可以在程序的任何地方调用。这消除了一遍又一遍写相同的代码的需要。这将会使程序员能够将一个大程序分成若干小的，可管理的函数功能。

除了内置函数，VBA允许我们编写用户定义的函数，以及和语句都应写Function和End Function之间

Sub过程

子程序的工作类似函数，Sub过程一般无返回值，函数可能会或可能不会返回一个值。 Sub过程可以不调用关键字。子过程总是使用Sub和End Sub语句括起来。

VBA宏注释 - VBA教程

VBA注释

注释是用来记录程序逻辑和用户信息与其他程序员可以在相同的代码无缝协同工作打好基础。

它可包括开发，修改的信息，例如，它也可以包括引入作为逻辑。注释被解释执行时忽略。

在VBA中的注释用两种方法来表示。

1.是用单引号(?)开始的任何语句都被当作注释。下面是一个例子：

```
' This Script is invoked after successful login
' Written by : YiiBai
' Return Value : True / False
```

2.以关键字“REM”开头的任何声明。下面是一个例子：

```
REM This Script is written to Validate the Entered Input
REM Modified by  : Yiibai/user_a
```

VBA消息框 - VBA教程

VBA消息框

MsgBox函数显示一个消息框，并等待用户点击一个按钮，然后根据用户点击该按钮的动作执行。

语法

```
MsgBox(prompt[,buttons][,title][,helpfile,context])
```

参数说明

- Prompt - 必需的参数。这显示在对话框中的消息的字符串。 prompt 最大长度大约是1024个字符。如果消息扩展到多行，那么可以单独使用回车符(CHR(13))或每行之间的换行符(CHR(10))。
- buttons - 一个可选的参数。数值表达式，用于指定按钮的类型来显示，图标样式使用，默认按钮的标识以及消息框的样式。如果留空，对于按钮的默认值是0。
- Title - 一个可选的参数。在对话框的标题栏中显示的字符串表达式。如果标题为空，应用程序的名称被放置在此标题栏中。
- helpfile - 一个可选的参数。标识帮助文件中的字符串表达式使用提供的对话框中的上下文相关帮助。
- context - 一个可选的参数。数值表达式，用于标识由帮助文件的作者指定给适当的帮助主题的上下文编号。如果上下文中提供帮助文件，此项还必须提供。

按钮参数可以采用任何的下列值：

- 0 vbOKOnly只显示OK按钮。
- 1 vbOKCancel显示确定和取消按钮。
- 2 vbAbortRetryIgnore显示放弃，重试和忽略按钮。
- 3 vbYesNoCancel Displays Yes, No, and Cancel buttons.
- 4 vbYesNoCancel显示是，否和取消按钮。
- 5 vbRetryCancel显示重试和取消按钮。

- 16 vbCritical显示关键信息的图标。
- 32 vbQuestion显示警告查询图标。
- 48 vbExclamation显示一条警告信息图标。
- 64 vbInformation显示信息消息图标。
- 0 vbDefaultButton1第一个按钮是默认的。
- 256 vbDefaultButton2第二个按钮是默认的。
- 512 vbDefaultButton3第三个按钮是默认的。
- 768 vbDefaultButton4第四个按钮是默认的。
- 0 vbApplicationModal应用模式。当前应用程序将无法正常工作，直到用户响应消息框。
- 4096 vbSystemModal系统模式。所有的应用程序将无法正常工作，直到用户响应消息框。

上面的值是逻辑上划分为四组：第一组(0-5)表示按钮被显示在消息框中。第二组(16，32，48，64)描述的图标的sytle要被显示，第三组(0，256，512，768)指示哪些键必须是缺省值，第四组值(0，4096)确定该消息框的样式。

返回值

MsgBox函数可以返回使用，我们将能够识别此按钮，用户在消息框中单击了下列值之一。

- 1 - vbOK - "确定"被点击
- 2 - vbCancel - "取消"被点击
- 3 - vbAbort - "中止"被点击
- 4 - vbRetry - "重试"被点击
- 5 - vbIgnore - "忽略"被点击
- 6 - vbYes - "是"被点击
- 7 - vbNo - "否"被点击

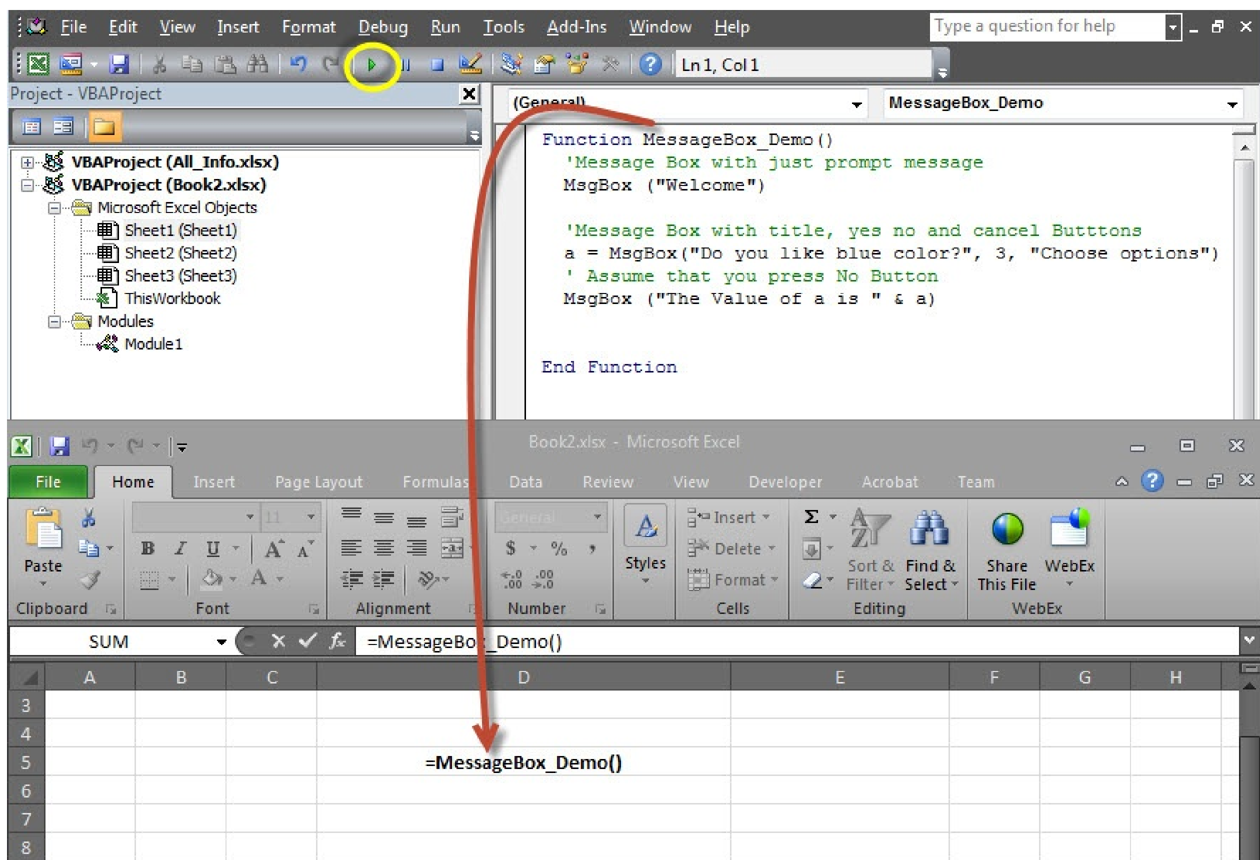
例子

```
Function MessageBox_Demo()
  'Message Box with just prompt message
  MsgBox("Welcome")

  'Message Box with title, yes no and cancel Butttons
  a = MsgBox("Do you like blue color?",3,"Choose options")
  ' Assume that you press No Button
  msgbox ("The Value of a is " & a)
End Function
```

输出

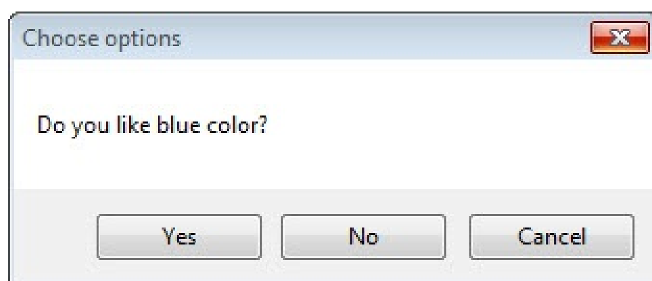
1.上述功能可以通过点击VBA窗口“运行”按钮，或通过调用Excel工作表函数，如下图所示执行。



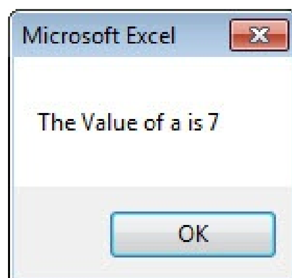
2.一个简单的消息框将显示一条消息，“Welcome”和“OK”按钮：



3.单击确定后， 另一个对话框将显示一条消息，“yes, no, 和 cancel”按钮。



4.单击取消按钮键的值之后（7）被存储为一个整数，如下所示显示消息框给用户。使用该值，我们就能够知道哪个按钮用户点击。



VBA输入框 - VBA教程

什么是输入框？

InputBox函数帮助用户从用户得到值。输入值时，如果用户点击OK按钮或键盘上按下ENTER后，InputBox函数将返回在文本框中的文本。如果用户点击取消按钮，该函数会返回一个空字符串("")。

语法

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])
```

参数说明：

- Prompt - 必需的参数。这显示在对话框中的消息的字符串。 prompt 最大长度大约是1024个字符。如果消息扩展到多行，那么可以单独使用回车符(CHR(13))或每行之间的换行符(CHR(10))。
- Title - 一个可选的参数。在对话框的标题栏中显示的字符串表达式。如果标题为空，应用程序的名称被放置在标题栏中。
- Default - 一个可选的参数。显示在文本框中的默认文本
- XPos - 一个可选的参数。X轴的位置，表示从屏幕的左侧水平提示距离。如果留空，输入框水平居中。
- YPos - 一个可选的参数。Y轴的位置，表示从画面垂直方向的左侧的提示距离。如果留空，输入框垂直居中。
- helpfile - 一个可选的参数。标识帮助文件中的字符串表达式使用提供的对话框中的上下文相关帮助。
- context - 一个可选的参数。数值表达式，用于标识由帮助文件的作者指定给适当的帮助主题的上下文编号。如果上下文中提供的，帮助文件还必须提供。

例子

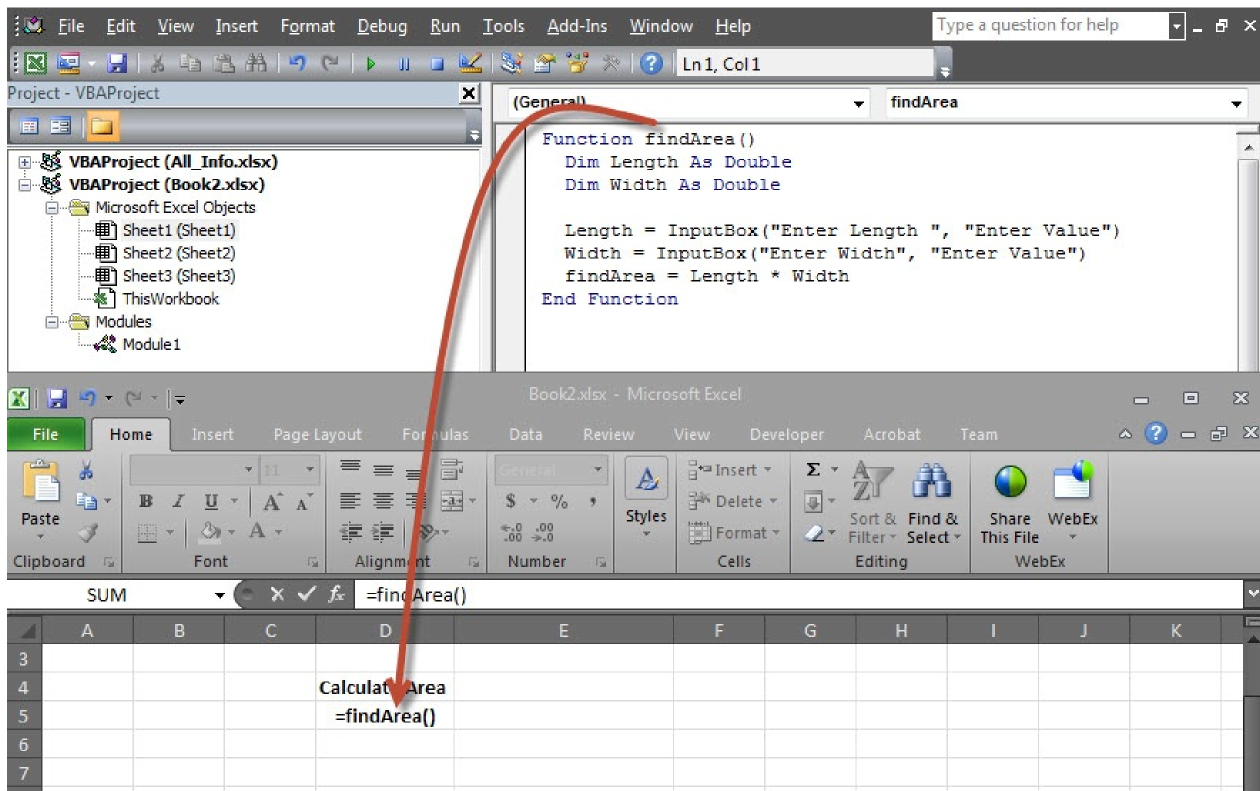
我们将通过从用户获得的值在运行时用的两个输入框(一个长度和一个用于宽度)的帮助下计算的矩形的面积

```
Function findArea()
  Dim Length As Double
  Dim Width As Double

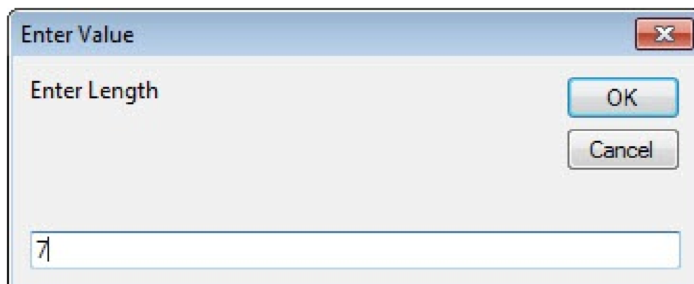
  Length = InputBox("Enter Length ", "Enter a Number")
  Width = InputBox("Enter Width", "Enter a Number")
  findArea = Length * Width
End Function
```

输出

1.执行相同，我们需要调用使用函数名，然后按下面输入如图所示。



2.在执行时，第一个输入框(长度)的显示和用户输入一个值在输入框中。



3.进入的第一个值之后，第二输入框(宽度)被显示给用户。

Enter Value

Enter Width

OK

Cancel

4

4.在进入第二数量并点击OK按钮，该区域被显示给用户，如下所示。

B	C	D	E
		Calculate Area	
		28	

VBA变量 - VBA教程

变量是用来存放可以在脚本执行过程中改变的值命名的存储位置。下面是命名变量的基本规则。下面所列的是用于命名一个变量的规则。

- 必须使用一个字母作为第一个字符。
- 不能使用空格，句号(.)，感叹号(!)，或字符@, &, $, #在变量名称中。
- 名称不能超过255个字符。
- 不能使用Visual Basic保留关键字作为变量名。

语法

在VBA中，我们需要在使用之前声明变量。

```
Dim <<variable_name>> As <<variable_type>>
```

数据类型

有许多的VBA的数据类型，它可以非常分为两大类，即数字和非数字数据类型。

数字数据类型

下表显示的数值数据类型和值的允许范围。

类型	值范围
Byte	0 - 255
Integer	-32,768 - 32,767
Long	-2,147,483,648 - 2,147,483,648
Single	-3.402823E+38 ~ -1.401298E-45 为负值 1.401298E-45 ~ 3.402823E+38 为正值
Double	-1.79769313486232e+308 ~ -4.94065645841247E-324 为负值 4.94065645841247E-324 ~ 1.79769313486232e+308 为正值
Currency	-922,337,203,685,477.5808 to 922,337,203,685,477.5807
Decimal	+/- 79,228,162,514,264,337,593,543,950,335 if no decimal is use +/- 7.9228162514264337593543950335 (28 decimal places).

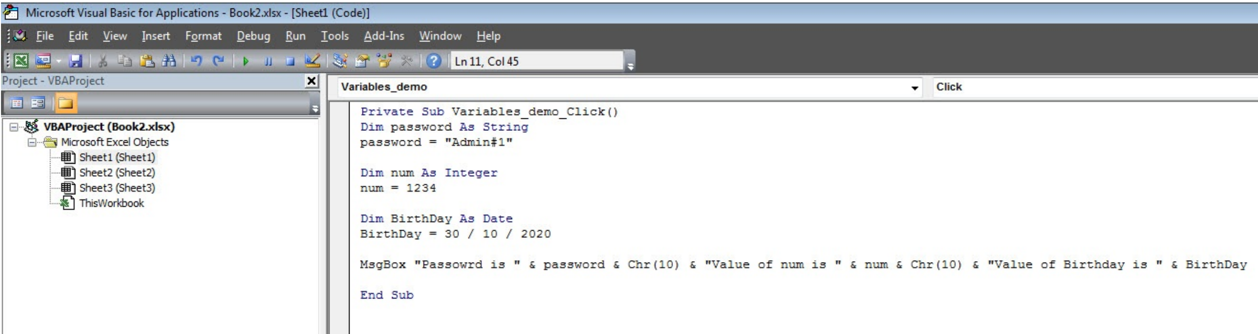
非数字数据类型

下表显示了非数值数据类型和值的允许范围。

类型	值范围
String(fixed length)	1 ~ 65,400 字符
String(variable length)	0 ~ 2 十亿个字符
Date	1月 1, 100 到12月 31, 9999
Boolean	True 或False
Object	任何嵌入对象
Variant(numeric)	任何Double值一样大
Variant(text)	同为可变长度的字符串

例子

我们创建一个按钮，并将其命名为“Variables_demo”用来演示使用变量。




```
Private Sub Variables_demo_Click()
 Dim password As String
 password = "Admin#1"

 Dim num As Integer
 num = 1234

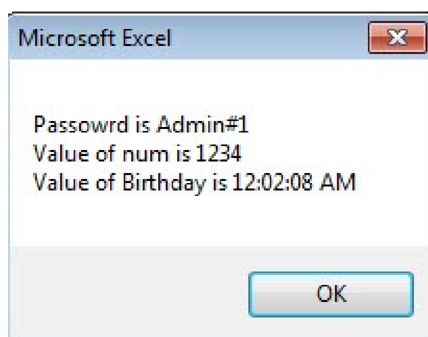
 Dim BirthDay As Date
 BirthDay = 30 / 10 / 2020

 MsgBox "Passowrd is " & password & Chr(10) & "Value of num is " & num & Chr(10) & "Value

End Sub
```

输出

时执行该脚本，则输出将如下所示。


Microsoft Excel
Passowrd is Admin#1
Value of num is 1234
Value of Birthday is 12:02:08 AM
OK

VBA常量 - VBA教程

常量是用来存放那些不能在脚本执行期间更改的值命名的存储位置。如果用户试图更改一个恒定值，该脚本执行出现一个错误并结束。常量声明的方式和变量声明相同。

下面是用于命名一个常量的规则。

- 必须使用一个字母作为第一个字符。
- 不能使用空格，句号(.)，感叹号(!)，或字符@, &, $, #在名称中。
- 名称不能超过255个字符。
- 不能使用Visual Basic保留关键字作为变量名。

语法

在VBA中，我们需要的值赋给声明的常量。如果我们试着改变常量的值错误会被抛出。

```
Const <<constant_name>> As <<constant_type>> = <<constant_value>>
```

例子

我们将创建一个按钮“Constant_demo”来演示如何使用常数。

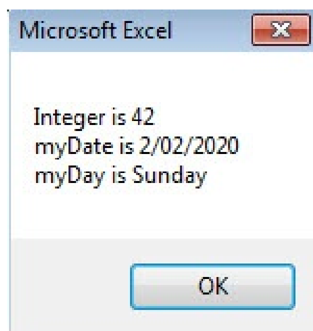
```
Private Sub Constant_demo_Click()
  Const MyInteger As Integer = 42
  Const myDate As Date = #2/2/2020#
  Const myDay As String = "Sunday"

  MsgBox "Integer is " & MyInteger & Chr(10) & "myDate is " & myDate & Chr(10) & "myDay i

End Sub
```

输出

在执行该脚本，如下所示，输出将被显示。


Microsoft Excel
Integer is 42
myDate is 2/02/2020
myDay is Sunday
OK

VBA运算符 - VBA教程

什么是运算符？

简单的回答可以利用公式4 + 5等于9，在这里，4和5被称为操作数，+被称为运算符给出。VBA支持以下类型的操作：

- 算术运算符
- 比较操作符
- 逻辑(或关系)操作符
- 连接操作

算术运算符

有以下是VBA支持的算术运算符：

假设变量A=5和变量B=10，则：

查看例子

运算符	描述	例子
+	两个操作数相加	A + B = 15
-	第一个操作数减去第二个操作数	A - B = -5
*	两个操作相乘	A * B = 50
/	通过分子除以分母	B / A = 2
%	模运算和整数相除后的余	B MOD A = 0
^	求幂运算符	B ^ A = 100000

比较运算符

以下是VBA支持的比较运算符：

假设变量A=10和变量B=20，则：

查看例子

运算符	描述	例子
==	检查，如果两个操作数的值是否相等，如果是，则条件变为true。	(A == B) 为 False.
<>	检查，如果两个操作数的值是否相等，如果值不相等，则条件变为true。	(A <> B) 为 True.
>	检查，如果左操作数的值大于右操作数的值，如果是的话那么条件为true。	(A > B) 为 False.
<	检查，如果左操作数的值小于右操作数的值，如果是的话那么条件为true。	(A < B) 为 True.
>=	检查，如果左边的操作数的值大于或等于右操作数的值，如果是，则条件变为true。	(A >= B) 为 False.
<=	检查，如果左边的操作数的值小于或等于右操作数的值，如果是，则条件变为true。	(A <= B) 为 True.

逻辑运算符：

以下是VBA支持的逻辑运算符：

假设变量A=10和变量B=0，则：

显示例子

运算符	描述	例子
AND	所谓逻辑与运算符。如果两个条件都为真则表达式为true。	a<>0 AND b<>0 is False.
OR	所谓逻辑OR运算符。如果有两个条件都为真则条件成立。	a<>0 OR b<>0 is true.
NOT	所谓逻辑非运算符。使用反转操作数的逻辑状态。如果条件为真，则逻辑非运算符将返回false。	NOT(a<>0 OR b<>0) is false.
XOR	所谓逻辑排除。这是不和OR运算符的结合。如果一个，只有一个，表达式的计算结果为真，结果为true。	(a<>0 XOR b<>0) is false.

串联运算符

以下是VBA支持级联运算符：

假设变量A=5和变量B=10，则：

显示例子

运算符	描述	例子
+	添加两个值的变量值数值	A + B = 15
&	连接两个值	A & B = 510

假设变量A=“Microsoft”和变量B =“VBScript”，则：

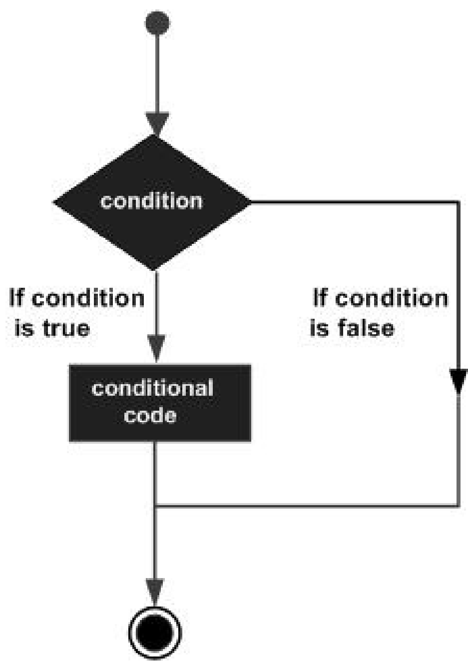
运算符	描述	例子
+	连接两个值	A + B = MicrosoftVBScript
&	连接两个值	A & B = MicrosoftVBScript

注：连接操作，可用于数字和字符串。输出取决于上下文，如果变量持有数值或字符串值。

VBA决策 - VBA教程

决策允许程序员控制脚本的执行流程或它的部分之一。执行是通过一个或多个条件语句的约束。

以下是在大多数编程语言中的一个典型的决策结构的一般形式：



VBA提供了以下几种类型的决策语句。点击以下链接查看的详细资料。

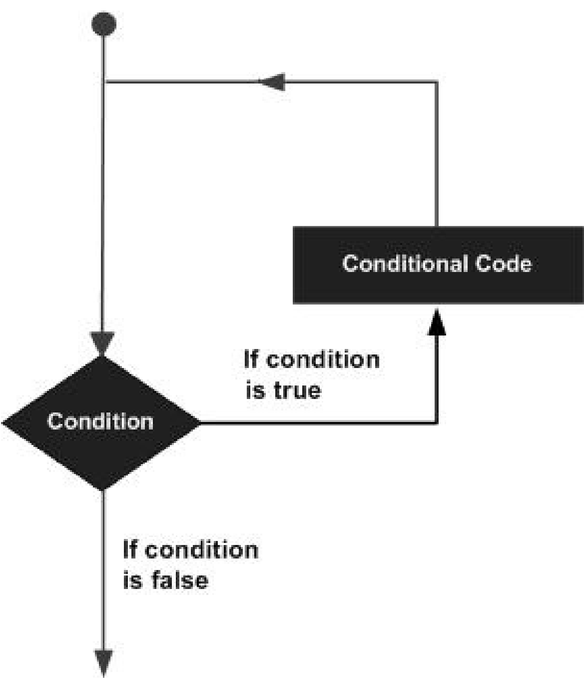
语句	描述
if 语句	if语句包含一个布尔表达式后跟一个或多个语句。
if..else 语句	if else语句包含一个布尔表达式后跟一个或多个语句。 如果条件为真，if语句执行。如果条件是假的，脚本的else部分被执行
if...elseif..else 语句	if语句后面跟着一个或多个elseif的声明，即由布尔表达式，然后跟着一个可选的else语句， 其中当所有的条件为假时 else语句部分执行。
内嵌 if 语句	if 或 elseif 语句中可以加入另一个 if 或 elseif 声明。
switch statement	switch语句允许一个变量， 为防止值列表相等进行测试。

VBA循环 - VBA教程

可能有一种情况，需要执行代码块多次。在一般情况下，语句顺序执行：在一个函数的第一条语句，首先执行，然后是第二个...等等。

编程语言提供了多种控制结构，使更复杂的执行路径。

循环语句可以执行语句的语句多次或多组，下面是VBA循环的一般语句。



VBA提供循环以下类型的处理循环的要求。点击以下链接查看其详细信息。

循环类型	描述
for 循环	执行语句多次序列，简写形式用于管理循环变量的代码。
for ..each 循环	这执行，如果有组的至少一种元素并且重申为在一组中的每个元素。
while..wend 循环	这在执行循环体之前测试条件。
do..while 循环	do..while语句只要条件为真时执行。（即）循环重复，直到条件为False。
do..until 循环	do..until语句执行直到条件为False。（即）循环应重复，只要条件为真。

循环控制语句：

循环控制语句改变其正常的顺序执行。当执行离开了循环范围，在循环中的所有剩余语句不执行。

VBA支持下列控制语句。点击以下链接查看其详细信息。

控制语句	描述
Exit For 语句	终止 for 循环语句并将执行的语句，将立即循环体的下面语句
Exit Do 语句	终止do while语句并将执行循环紧随其后的语句

VBA字符串 - VBA教程

字符串是字符，这可以由字母或数字或特殊字符或所有的序列。一个变量的值如果使用双引号""，那么它会被认为是一个字符串。

语法：

```
variablename = "string"
```

例如：

```
str1 = "string"    ' Only Alphabets
str2 = "132.45"    ' Only Numbers
str3 = "!@#$;*"  ' Only Special Characters
Str4 = "Asc23@#"  ' Has all the above
```

字符串函数：

预定义VBA字符串函数，这有助于开发人员使用字符串能非常有效的工作。下面是在 VBA 支持字符串的方法。请点击的方法，每个人都应该知道的细节。

函数名称	描述
InStr	返回指定字符串的第一次出现。搜索从发生左向右。
InstrRev	返回指定字符串的第一次出现。搜索发生从右到左。
Lcase	返回指定字符串的小写。
Ucase	返回指定字符串的大写。
Left	返回字符从字符串的左侧的特定数目。
Right	返回字符从字符串的右侧的特定数目。
Mid	返回从基于指定参数的字符串的字符的特定数目。
Ltrim	返回指定的字符串的左侧去除空格之后的字符串。
Rtrim	返回在右侧的指定字符串的去除空格之后的一个字符串。
Trim	返回删除这两个开头和结尾空格后的字符串值。
Len	返回给定字符串的长度。
Replace	返回字符串替换字符串后的字符串。
Space	填补了字符串的空格指定的数目。
StrComp	返回比较两个指定的字符串后的整数值。
String	返回具有指定字符的指定次数的字符串。
StrReverse	返回反转给定字符串的字符序轮机后的字符串。

VBA日期时间函数 - VBA教程

VBScript的日期和时间函数帮助开发人员转换日期和时间，转换另一种格式或表达的适合特定条件格式的日期或时间值。

日期函数

函数	描述
Date	一个函数，返回当前系统日期
CDate	一个函数，一个给定的输入转换为日期
DateAdd	一个函数，返回已添加日期到其指定的时间间隔
DateDiff	一个函数，返回在两个时间段的差异（差值）
DatePart	一个函数，返回给定输入日期值的指定部分
DateSerial	一个函数，返回一个有效日期为特定年份，月份和日期
FormatDateTime	一个函数，它将格式化基于提供的参数日期
IsDate	一个函数，返回一个布尔值，是否提供的参数是一个日期
Day	一个函数，返回1到31之间的整数，表示指定日期的当天
Month	一个函数，返回1到12之间的整数，表示指定日期的月份
Year	一个函数，返回一个整数，表示指定日期的年份
MonthName	一个函数，返回特定月份名称为指定日期
WeekDay	一个函数，返回一个整数（1～7），表示星期几指定一天。
WeekDayName	一个函数，返回星期名称指定的一天。

时间函数

函数	描述
Now	一个函数，返回当前系统日期和时间
Hour	一个函数返回介于0和23，整数表示的给定时间的小时部分
Minute	一个函数，返回介于0和59，整数表示分钟的指定时间部分
Second	一个介于0和59功能，返回整数表示的给定时间的秒数部分
Time	一个函数，返回当前系统时间
Timer	一个函数，返回秒和毫秒的数量自12:00 AM
TimeSerial	一个函数，将特定的输入返回时间的小时，分钟和秒
TimeValue	一个函数，把输入字符串转换为时间格式

VBA数组 - VBA教程

什么是数组？

我们非常清楚地知道，一个变量是一个容器来存储值。有时开发者在一个位置，在一个单一的变量一次持有多个值。当一系列值被存储在一个单独的变量，那么它被称为数组变量。

声明数组

数组声明以其它变量的方式同样，只是数组变量的声明使用圆括号声明。在下面的例子中，数组大小在括号中指定。

```
'Method 1 : Using Dim
Dim arr1()    'Without Size

'Method 2 : Mentioning the Size
Dim arr2(5)  'Declared with size of 5

'Method 3 : using 'Array' Parameter
Dim arr3
arr3 = Array("apple","Orange","Grapes")
```

1. 虽然，数组大小显示为5，它可以容纳6个值作为数组索引从零开始。
2. 数组索引不能为负数。
3. VBScript数组可以存储任何类型的变量数组。因此，一个阵列可以存储的整数，串或字符在一个单一的数组变量。

赋值数组

数值通过指定数组索引值对值中的每一个将被分配被分配到阵列。它可以是一个字符串。

例子：

添加一个按钮，并添加以下功能

```
Private Sub Constant_demo_Click()
  Dim arr(5)
  arr(0) = "1"           'Number as String
  arr(1) = "VBScript"    'String
  arr(2) = 100            'Number
  arr(3) = 2.45            'Decimal Number
  arr(4) = #10/07/2013#  'Date
  arr(5) = #12.45 PM#    'Time

  msgbox("Value stored in Array index 0 : " & arr(0))
  msgbox("Value stored in Array index 1 : " & arr(1))
  msgbox("Value stored in Array index 2 : " & arr(2))
  msgbox("Value stored in Array index 3 : " & arr(3))
  msgbox("Value stored in Array index 4 : " & arr(4))
  msgbox("Value stored in Array index 5 : " & arr(5))
End Sub
```

当执行函数输出如下所示：

```
Value stored in Array index 0 : 1
Value stored in Array index 1 : VBScript
Value stored in Array index 2 : 100
Value stored in Array index 3 : 2.45
Value stored in Array index 4 : 7/10/2013
Value stored in Array index 5 : 12:45:00 PM
```

多维数组

数组并不仅仅局限于单一的维度，最多可有60维度。最常用的是二维数组。

例子：

在下面的例子中，一个多维阵列具有3行和4列声明。

```
Private Sub Constant_demo_Click()
  Dim arr(2,3) as Variant    ' Which has 3 rows and 4 columns
  arr(0,0) = "Apple"
  arr(0,1) = "Orange"
  arr(0,2) = "Grapes"
  arr(0,3) = "pineapple"
  arr(1,0) = "cucumber"
  arr(1,1) = "beans"
  arr(1,2) = "carrot"
  arr(1,3) = "tomato"
  arr(2,0) = "potato"
  arr(2,1) = "sandwitch"
  arr(2,2) = "coffee"
  arr(2,3) = "nuts"

  msgbox("Value in Array index 0,1 : " &  arr(0,1))
  msgbox("Value in Array index 2,2 : " &  arr(2,2))

End Sub
```

当执行函数输出如下所示：

```
Value stored in Array index : 0 , 1 : Orange
Value stored in Array index : 2 , 2 : coffee
```

Redim 语句

ReDim语句用于声明动态数组变量并分配或重新分配存储空间。

```
ReDim [Preserve] varname(subscripts) [, varname(subscripts)]
```

- Preserve - 可选参数，用来当改变最后一维的大小来保存数据在现有的数组。
- varname - 必需的参数，它表示的变量，它应该遵循标准的变量命名约定的名称。
- subscripts - 必需的参数，它表示该数组大小。

例子

在下面的例子中一个数组已经被重新定义，在保存的值时该数组的现有大小被改变。

注：在调整大小的数组小于它最初的值，在消除元素的数据将会丢失。

```
Private Sub Constant_demo_Click()
  Dim a() as variant
  i=0
  redim a(5)
  a(0)="XYZ"
  a(1)=41.25
  a(2)=22

  REDIM PRESERVE a(7)
  For i=3 to 7
  a(i)= i
  Next

  'to Fetch the output
  For i=0 to ubound(a)
    Msgbox a(i)
  Next
End Sub
```

当执行函数输出如下所示：

```
XYZ
41.25
22
3
4
5
6
7
```

数组方法：

在VBScript中的各种内置函数，帮助开发者有效地处理数组。所有正在使用中一起选择数组方法在下面列出。请点击方法名详细了解。

函数	描述
LBound	此函数返回一个整数，对应于给定的数组中最小的下标。
UBound	此函数返回一个整数，对应于给定数组的最大下标。
Split	此函数返回包含值的指定数量的数组。分割后基于分隔符。
Join	此函数返回一个包含子字符串数组中的指定数量的字符串。这是Split 方法的完全相反的作用。
Filter	此函数返回零的数组包含字符串数组基于一个特定的过滤标准的子集。
IsArray	此函数返回一个布尔值，指示输入变量是否是一个数组。
Erase	此函数恢复所分配的内存为数组变量。

VBA定义函数 - VBA教程

什么是函数？

函数是一组可重用的代码，可以在程序的任何地方被调用。这消除了一遍又一遍写相同的代码的需要。这将使程序员将一个大程序分成若干小且易于管理的功能。

除了内置的功能，VBA允许我们编写的用户定义函数也是如此。本节将介绍如何编写在VBA中自己定义的函数。

函数定义

VBA函数可以有一个可选的return语句。如果想从一个函数返回一个值这是必需的。

例如，可以通过两个数字在一个函数，那么可以从函数希望返回在调用程序乘法。

注：函数可以返回由逗号作为分配给函数名本身就是一个数组分隔的多个值。

在我们使用一个函数之前，我们需要先定义特定函数。在VBA中定义函数的最常见的方法是通过使用 Function 关键字，随后是唯一的函数名称，并将其可以或可以不携带的参数的列表，并与一个 End Function 关键字结束，这表明该函数结束声明。基本语法如下所示：

语法

添加一个按钮，并添加以下功能

```
Function Functionname(parameter-list)
  statement 1
  statement 2
  statement 3
  .......
  statement n
End Function
```

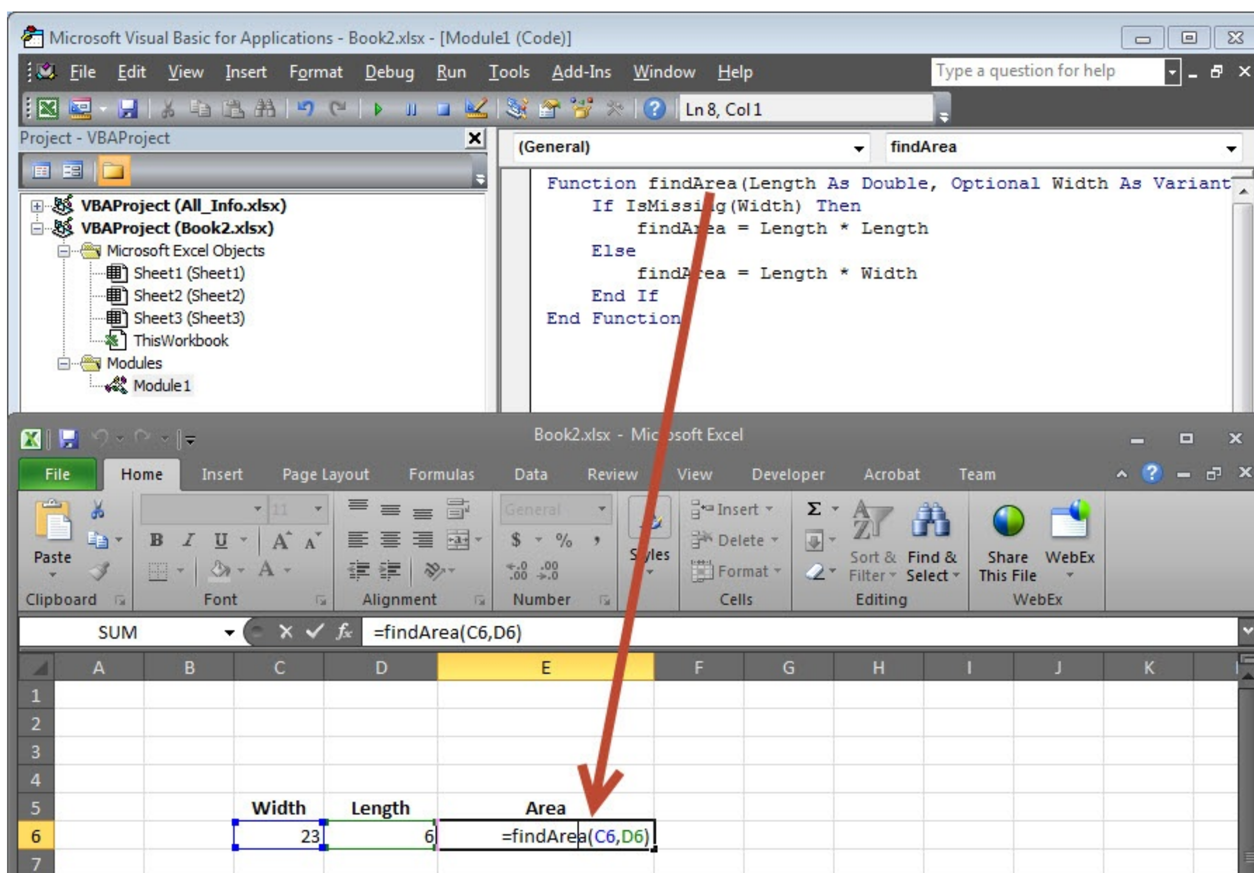
例子

添加以下函数返回面积。需要注意的是一个值/值可以连同函数名本身被返回。

```
Function findArea(Length As Double, Optional Width As Variant)
    If IsMissing(Width) Then
        findArea = Length * Length
    Else
        findArea = Length * Width
    End If
End Function
```

调用函数

调用一个函数，调用使用函数名称，如下所示：



VBA子过程 - VBA教程

子过程

Sub过程类似函数，但也有一些差别。

- 子过程没有返回值，同时函数可能会或可能不会返回值。
- 子过程调用可以不用关键字。
- 子过程总是在Sub和End Sub语句之间括起来部分。

例子：

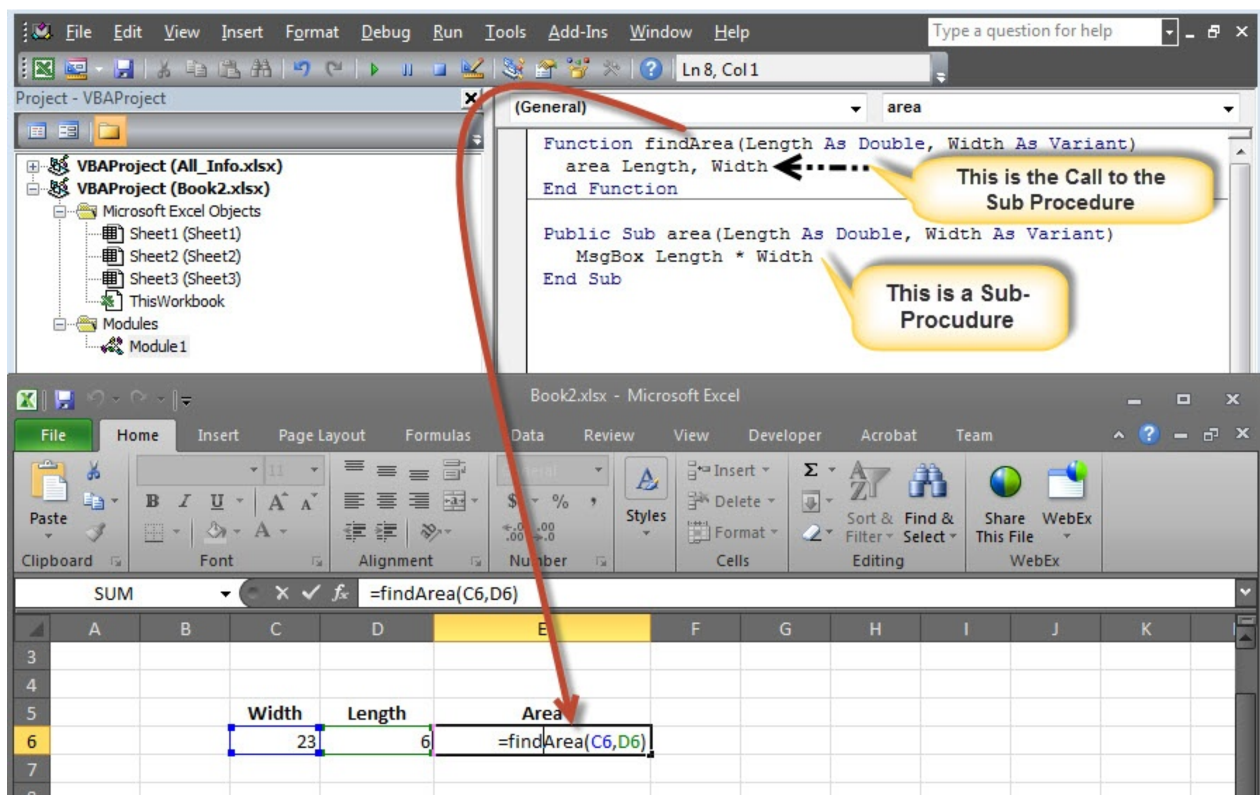
```
Sub Area(x As Double, y As Double)
    MsgBox x * y
End Sub
```

调用过程：

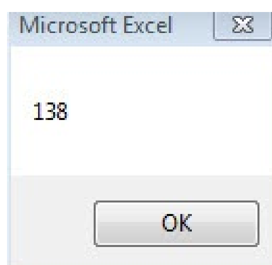
在脚本的某处调用程序，可以从一个函数调用。但不能够使用相同的方式，一个功能的子过程是没有返回值的。

```
Function findArea(Length As Double, Width As Variant)
    area Length, Width    ' To Calculate Area 'area' sub proc is called
End Function
```

1.现在可以调用函数只而不是子过程，如下图所示。



2.面积计算，仅在消息框中显示。



3.结果单元格显示为零面积值不是从函数返回。总之，不能直接从Excel工作表调用一个子过程。

Width	Length	Area
23	6	0

The Output is shown as ZERO as the sub procedure displays the message box and no value is returned from the function.

VBA事件 - VBA教程

VBA 事件

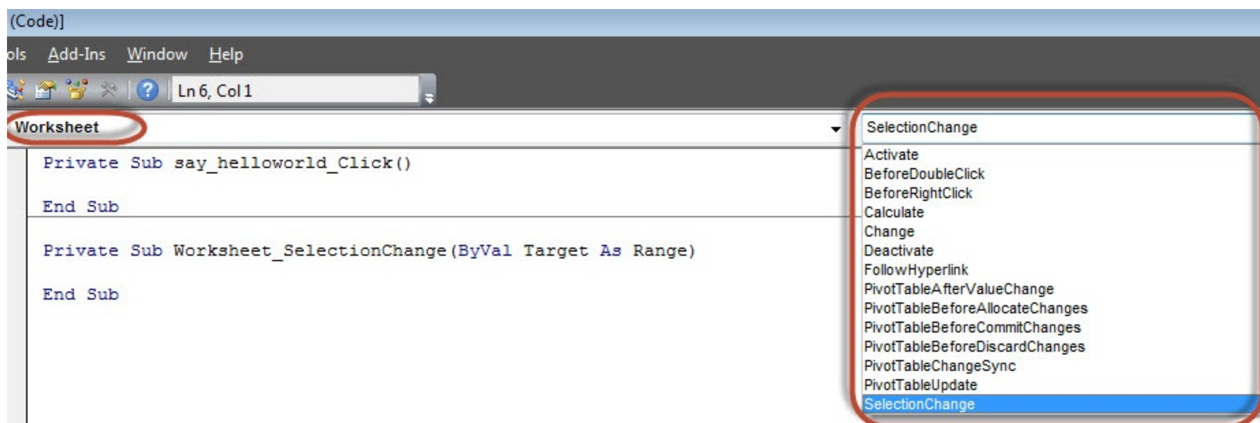
VBA，事件驱动编程时可以手动更改单元格值的单元格或单元格区域被触发。更改事件可能会使事情变得更容易，但可以很快结束了一个完全格式化的页面。有两种类型的事件。

- 工作表事件
- 工作簿活动

工作表事件

工作表事件被触发时，在工作表中有变化。表标签上执行右键单击，选择“view code”，然后粘贴代码创建的。

用户可以选择那些工作表中的每一个，并从下拉列表中选择“工作表”下去把所有支持工作表的事件列表。



下面是可以由用户添加的支持工作表的事件。

```
Private Sub Worksheet_Activate()
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
Private Sub Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)
Private Sub Worksheet_Calculate()
Private Sub Worksheet_Change(ByVal Target As Range)
Private Sub Worksheet_Deactivate()
Private Sub Worksheet_FollowHyperlink(ByVal Target As Hyperlink)
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
```

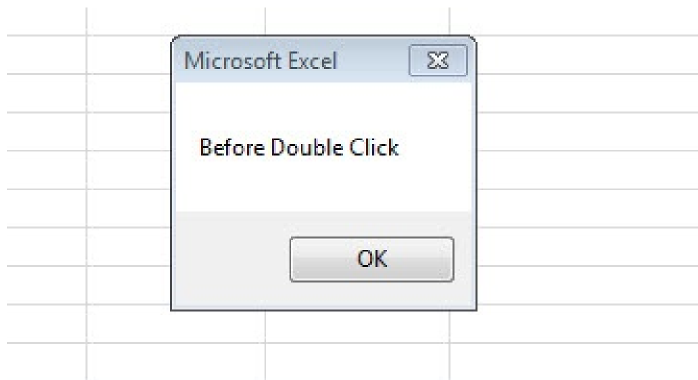
例子

只需要前双击显示一条消息。

```
Private Sub Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)
    MsgBox "Before Double Click"
End Sub
```

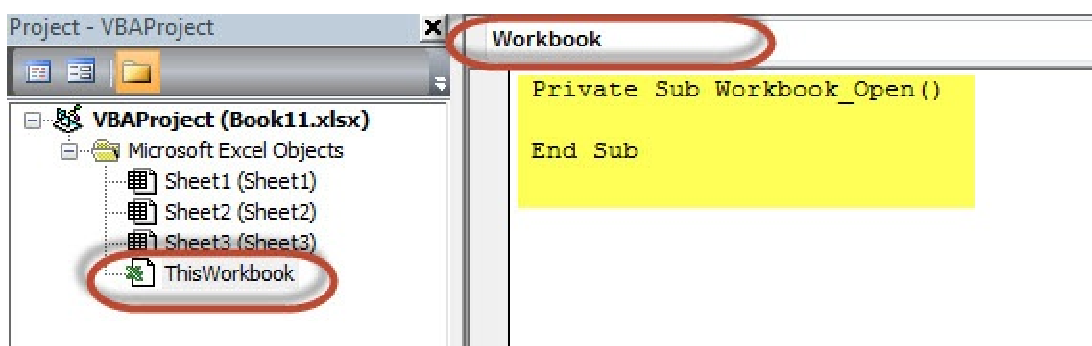
输出

当双击任意单元格，显示消息框给用户，如下所示。



工作簿活动

工作簿事件被触发时，有一个变化以对整个工作簿。可以通过选择“ThisWorkbook'和选择从下拉'workbook'，如下所示添加为工作簿的事件的代码。立即 Workbook_open 子过程显示给用户，如下所示。



下面是可以由用户添加的支持工作薄的事件。

```
Private Sub Workbook_AddinUninstall()
Private Sub Workbook_BeforeClose(Cancel As Boolean)
Private Sub Workbook_BeforePrint(Cancel As Boolean)
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)
Private Sub Workbook_Deactivate()
Private Sub Workbook_NewSheet(ByVal Sh As Object)
Private Sub Workbook_Open()
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, ByVal Target As Range, Ca
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, ByVal Target As Range, Can
Private Sub Workbook_SheetCalculate(ByVal Sh As Object)
Private Sub Workbook_SheetChange(ByVal Sh As Object, ByVal Target As Range)
Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
Private Sub Workbook_SheetFollowHyperlink(ByVal Sh As Object, ByVal Target As Hyperlink)
Private Sub Workbook_SheetSelectionChange(ByVal Sh As Object, ByVal Target As Range)
Private Sub Workbook_WindowActivate(ByVal Wn As Window)
Private Sub Workbook_WindowDeactivate(ByVal Wn As Window)
Private Sub Workbook_WindowResize(ByVal Wn As Window)
```

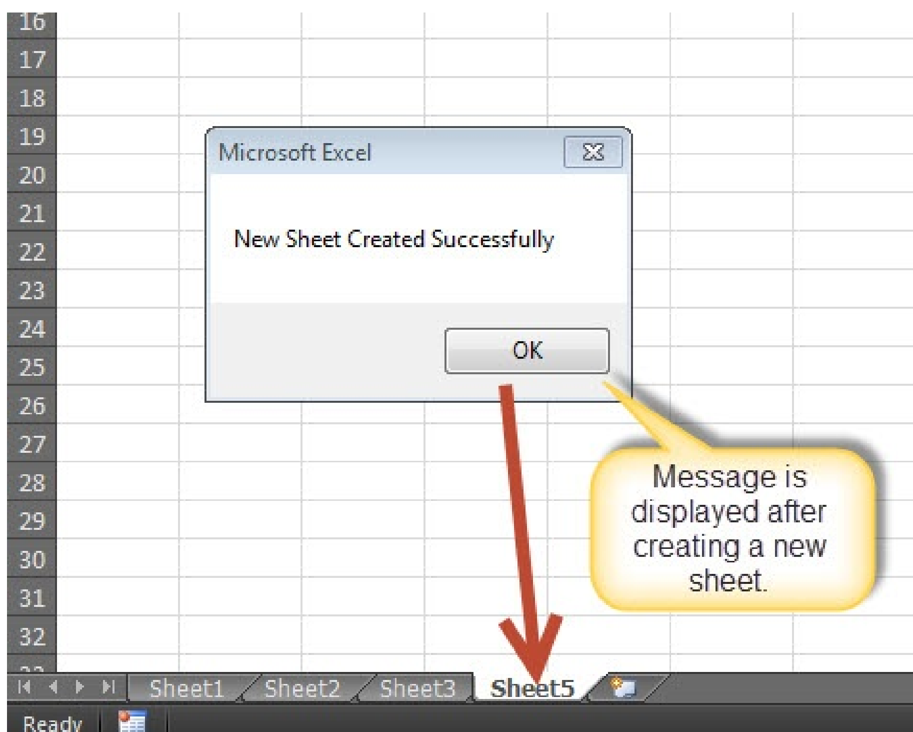
例子

只需要显示一条消息，一个新的工作表被成功创建，每当一个新表是创建的用户。

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    MsgBox "New Sheet Created Successfully"
End Sub
```

输出

创建一个新的 Excel工作表的消息显示给用户，如下所示。



VBA错误处理 - VBA教程

有三种类型的编程错误：(1)语法错误和(b)运行时错误(三)逻辑错误。

语法错误

语法错误，也被称为解析错误，发生在VBScript解释的时候。例如，下面的一行将导致一个语法错误，因为它缺少一个右括号：

```
Function ErrorHanlding_Demo()
dim x,y
x = "Yiibai"
y = Ucase(x
End Function
```

运行时错误

在执行过程中运行时错误，也被称为异常，发生解释后。

例如，下面的行导致运行时错误，因为这里的语法是正确的，但在运行时，它试图调用乘法，这是一个不存在的功能：

```
Function ErrorHanlding_Demo1()
  Dim x,y
  x = 10
  y = 20
  z = fnadd(x,y)
  a = fnmultiply(x,y)
End Function

Function fnadd(x,y)
      fnadd = x+y
End Function
```

逻辑错误

逻辑错误可能是最困难的类型的错误跟踪。这些错误不是一个语法或运行时错误的结果。相反，当犯了脚本逻辑的一个错误发生，没有得到所期望的结果。

无法抓到这些错误，因为这取决于你想要把什么样的程序执行在逻辑业务需求中。

例如，一个数字除以零或进入无限循环，写在脚本中。

Err对象

假设，如果有一个运行时错误，则执行停止通过显示错误信息。作为开发人员，如果想捕捉错误，那么可以使用Error对象。

示例

在下面的例子中，Err.Number 给出了错误号和 Err.Description 给出错误描述。

```
Err.Raise 6   ' Raise an overflow error.
MsgBox "Error # " & CStr(Err.Number) & " " & Err.Description
Err.Clear   ' Clear the error.
```

错误处理

VBA有一个错误处理例程，也可以用于禁用一个错误处理例程。如果没有On Error语句，发生的任何运行时错误是致命的：将显示一条错误消息，并且执行突然停止。

```
On Error { GoTo [ line | 0 | -1 ] | Resume Next }
```

关键字	描述
GoTo line	可允许在需要的行参数指定的行开始的错误处理例程。指定的行必须在相同的过程中On Error语句，否则将发生编译时错误。
GoTo 0	禁用启用的错误处理当前过程中，并重置为Nothing。
GoTo -1	在当前过程中禁用启用例外，它重置为Nothing。
Resume Next	指定在运行时发生错误时，控制转到该语句立即出现错误的语句之后，并继续从该点执行

示例

```
Public Sub OnErrorDemo()
   On Error GoTo ErrorHandler   ' Enable error-handling routine.
   Dim x, y, z As Integer
   x = 50
   y = 0
   z = x / y   ' Divide by ZERO Error Raises

   ErrorHandler:    ' Error-handling routine.
   Select Case Err.Number   ' Evaluate error number.
      Case 10   ' Divide by zero error
         MsgBox ("You attempted to divide by zero!")
      Case Else
         MsgBox "UNKNOWN ERROR  - Error# " & Err.Number & " : " & Err.Description
   End Select
   Resume Next
End Sub
```

VBA Excel对象 - VBA教程

什么是Excel对象？

当使用VBA编程，用户将处理的一些重要的对象。

- 应用对象
- 工作簿对象
- 工作表对象
- 对象范围

应用对象

Application对象包括以下

- 应用范围的设置和选项。
- 方法返回顶层对象，如ActiveCell，ActiveSheet等等。

示例

```
'Example 1 :
Set xlapp = CreateObject("Excel.Sheet")
xlapp.Application.Workbooks.Open "C:\test.xls"

'Example 2 :
Application.Windows("test.xls").Activate

'Example 3:
Application.ActiveCell.Font.Bold = True
```

工作簿对象

工作簿对象是工作簿集合的成员，并包含所有当前在Microsoft Excel中打开的工作簿对象。

例子

```
'Ex 1 : To close Workbooks
Workbooks.Close

'Ex 2 : To Add an Empty Work Book
Workbooks.Add

'Ex 3: To Open a Workbook
Workbooks.Open FileName:="Test.xls", ReadOnly:=True

'Ex : 4 - To Activate WorkBooks
Workbooks("Test.xls").Worksheets("Sheet1").Activate
```

工作表对象

工作表对象是工作表集合的成员，并包含一个工作簿中所有工作表对象。

例子

```
'Ex 1 : To make it Invisible
Worksheets(1).Visible = False

'Ex 2 : To protect an WorkSheet
Worksheets("Sheet1").Protect password:=strPassword, scenarios:=True
```

范围对象

范围对象表示一个单元，一排，一列，可以选择含有单元格的一个或多个连续块。

```
'Ex 1 : To Put a value in the cell A5
Worksheets("Sheet1").Range("A5").Value = "5235"

'Ex 2 : To put a value in range of Cells
Worksheets("Sheet1").Range("A1:A4").Value = 5
```

VBA文本文件 - VBA教程

VBA文本文件

我们可以读取Excel文件，并写入单元格中的内容到一个文本文件。这样一来，VBA允许用户使用文本文件的工作。我们可以测试文件使用工作的两种方法

- 文件系统对象
- 使用Write命令

使用文件系统对象(FSO)

正如其名称所说的，FSO对象帮助开发者使用驱动器，文件夹和文件的工作。在本节中，我们将讨论如何使用FSO。

对象类型	描述
Drive	驱动器是一个对象。包含的方法和属性，收集关于连接到系统的驱动器的信息
Drives	硬盘是一个集合。它提供了连接到系统，无论是物理或逻辑的驱动器的列表。
File	文件是一个对象。它包含的方法和属性，使开发人员能够创建，删除或移动文件。
Files	文件是一个集合。它提供了包含在文件夹内的所有文件的列表。
Folder	文件夹是一个对象。它提供的方法和属性，使开发人员能够创建，删除或移动文件夹。
Folders	文件夹是一个集合。它提供了一个文件夹内的所有文件夹列表。
TextStream	文本流是一个对象。它使开发人员能够读取和写入文本文件。

驱动器

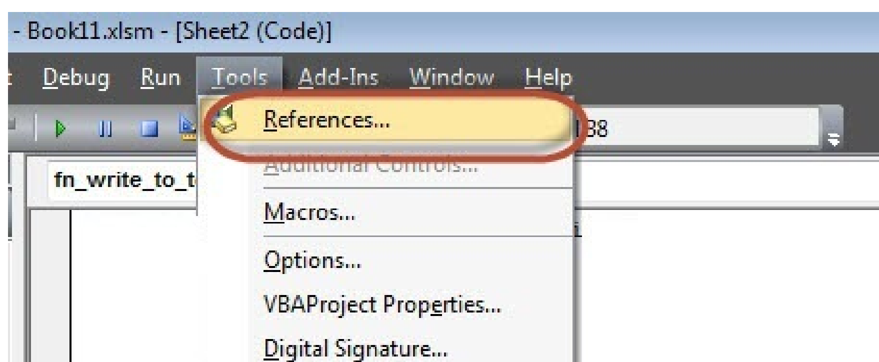
Drive是一个对象，它提供了访问特定的磁盘驱动器或网络共享的属性。以下属性是由驱动器对象支持：

- AvailableSpace
- DriveLetter
- DriveType

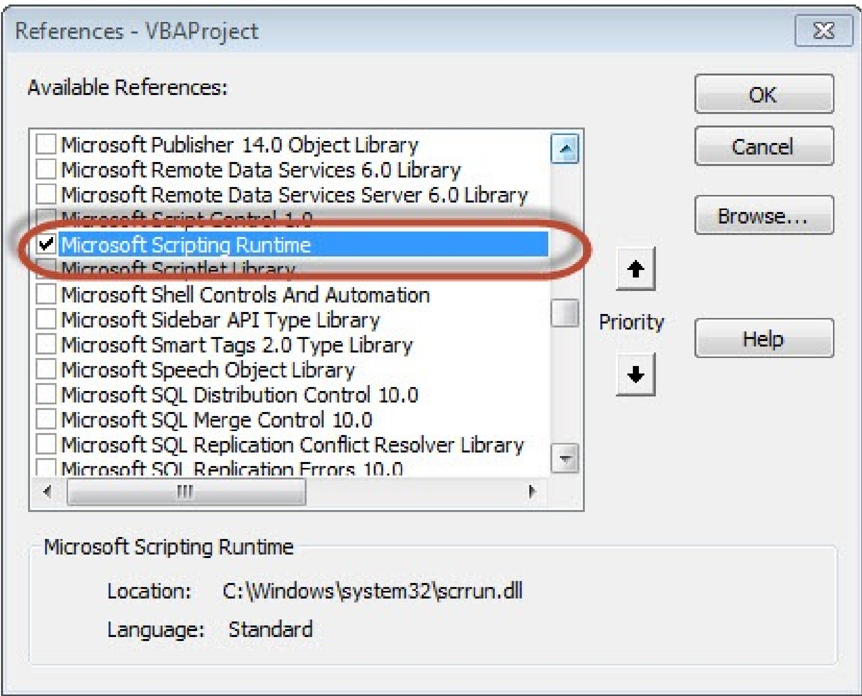
- FileSystem
- FreeSpace
- IsReady
- Path
- RootFolder
- SerialNumber
- ShareName
- TotalSize
- VolumeName

例子

第1步：在继续使用FSO脚本，我们应该使Microsoft脚本运行。做同样，导航到"Tools" >> "References"，如下图所示：



第2步：添加“Microsoft Scripting RunTime”，然后单击确定。



第3步：添加数据，将它写入一个文本文件，并添加一个命令按钮。

A1 | State

State	Prevalence	95% Confidence Interval
Alabama	33	(31.5, 34.4)
Alaska	25.7	(23.9, 27.5)
Arizona	26	(24.3, 27.8)
Arkansas	34.5	(32.7, 36.4)
California	25	(23.9, 26.0)
Colorado	20.5	(19.5, 21.4)
Connecticut	25.6	(24.3, 26.9)
Delaware	26.9	(25.2, 28.6)
District of Columbia	21.9	(19.8, 24.0)

Write to Text File

第4步：现在是写脚本的时候。

```
Private Sub fn_write_to_text_Click()
  Dim FilePath As String
  Dim CellData As String
  Dim LastCol As Long
  Dim LastRow As Long

  Dim fso As FileSystemObject
  Set fso = New FileSystemObject
  Dim stream As TextStream

  LastCol = ActiveSheet.UsedRange.Columns.Count
  LastRow = ActiveSheet.UsedRange.Rows.Count

  ' Create a TextStream.
  Set stream = fso.OpenTextFile("D:\Try\Support.log", ForWriting, True)

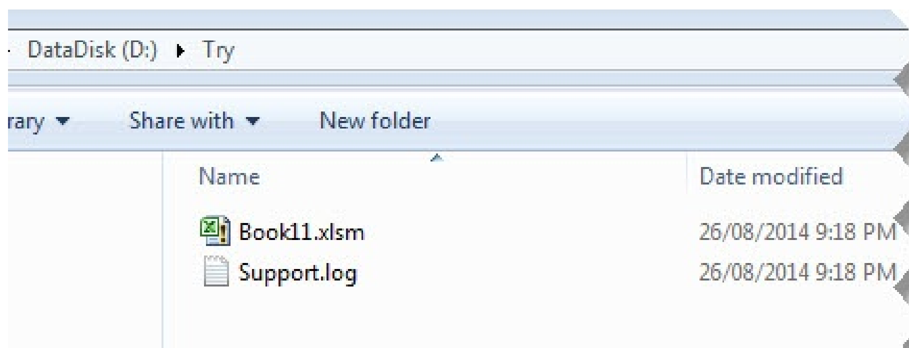
  CellData = ""

  For i = 1 To LastRow
      For j = 1 To LastCol
      CellData = Trim(ActiveCell(i, j).Value)
      stream.WriteLine "The Value at location (" & i & "," & j & ")" & CellData
      Next j
  Next i

  stream.Close
  MsgBox ("Job Done")
End Sub
```

输入

当执行脚本，请确保将光标放在工作表的第一个单元格。如在下面创建Support.log文件"D:\Try"：



该文件的内容也被显示如下：

```
1  The Value at location (1,1)State
2  The Value at location (1,2)Prevalence
3  The Value at location (1,3)95% Confidence Interval
4  The Value at location (2,1)Alabama
5  The Value at location (2,2)33
6  The Value at location (2,3)(31.5, 34.4)
7  The Value at location (3,1)Alaska
8  The Value at location (3,2)25.7
9  The Value at location (3,3)(23.9, 27.5)
10 The Value at location (4,1)Arizona
11 The Value at location (4,2)26
12 The Value at location (4,3)(24.3, 27.8)
13 The Value at location (5,1)Arkansas
14 The Value at location (5,2)34.5
15 The Value at location (5,3)(32.7, 36.4)
16 The Value at location (6,1)California
17 The Value at location (6,2)25
18 The Value at location (6,3)(23.9, 26.0)
19 The Value at location (7,1)Colorado
20 The Value at location (7,2)20.5
21 The Value at location (7,3)(19.5, 21.4)
22 The Value at location (8,1)Connecticut
23 The Value at location (8,2)25.6
24 The Value at location (8,3)(24.3, 26.9)
25 The Value at location (9,1)Delaware
26 The Value at location (9,2)26.9
27 The Value at location (9,3)(25.2, 28.6)
28 The Value at location (10,1)District of Columbia
29 The Value at location (10,2)21.9
30 The Value at location (10,3)(19.8, 24.0)
31
```

使用写命令

不像FSO，我们不需要添加任何引用，但是不能够正常工作的驱动器，文件和文件夹。能够只流添加到文本文件中。

例子

```
Private Sub fn_write_to_text_Click()
  Dim FilePath As String
  Dim CellData As String
  Dim LastCol As Long
  Dim LastRow As Long

  LastCol = ActiveSheet.UsedRange.Columns.Count
  LastRow = ActiveSheet.UsedRange.Rows.Count

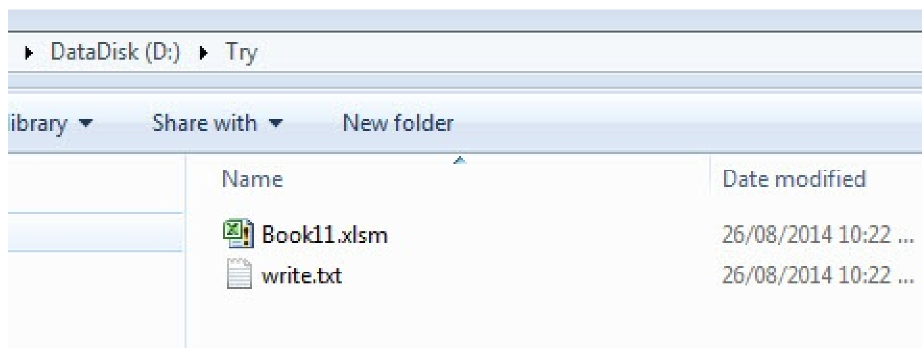
  FilePath = "D:\Try\write.txt"
  Open FilePath For Output As #2

  CellData = ""
  For i = 1 To LastRow
      For j = 1 To LastCol
      CellData = "The Value at location (" & i & "," & j & ")" & Trim(ActiveCell(i, j).Va
      Write #2, CellData
      Next j
  Next i

  Close #2
  MsgBox ("Job Done")
End Sub
```

输出

如下图所示当执行脚本时，在“D:\Try”创建“write.txt”文件。



该文件的内容也被显示如下：

```
1  The Value at location (1,1)State
2  The Value at location (1,2)Prevalence
3  The Value at location (1,3)95% Confidence Interval
4  The Value at location (2,1)Alabama
5  The Value at location (2,2)33
6  The Value at location (2,3)(31.5, 34.4)
7  The Value at location (3,1)Alaska
8  The Value at location (3,2)25.7
9  The Value at location (3,3)(23.9, 27.5)
10 The Value at location (4,1)Arizona
11 The Value at location (4,2)26
12 The Value at location (4,3)(24.3, 27.8)
13 The Value at location (5,1)Arkansas
14 The Value at location (5,2)34.5
15 The Value at location (5,3)(32.7, 36.4)
16 The Value at location (6,1)California
17 The Value at location (6,2)25
18 The Value at location (6,3)(23.9, 26.0)
19 The Value at location (7,1)Colorado
20 The Value at location (7,2)20.5
21 The Value at location (7,3)(19.5, 21.4)
22 The Value at location (8,1)Connecticut
23 The Value at location (8,2)25.6
24 The Value at location (8,3)(24.3, 26.9)
25 The Value at location (9,1)Delaware
26 The Value at location (9,2)26.9
27 The Value at location (9,3)(25.2, 28.6)
28 The Value at location (10,1)District of Columbia
29 The Value at location (10,2)21.9
30 The Value at location (10,3)(19.8, 24.0)
```

VBA图表编程 - VBA教程

VBA - 图表编程

使用VBA，就可以做到生成基于一定的标准图表。让我们来看看它的一个例子。

步骤**1**：首先输入针对图表有生成的数据。

Q23

	A	B
1	Year	Fuel Usage in Million Cubic Meters
2	1980	185.5
3	1990	214.1
4	2000	467.34
5	2010	1023.77
6		

第**2**步：让我们创建3个按钮中的一个来生成柱状图，饼图，柱形图。



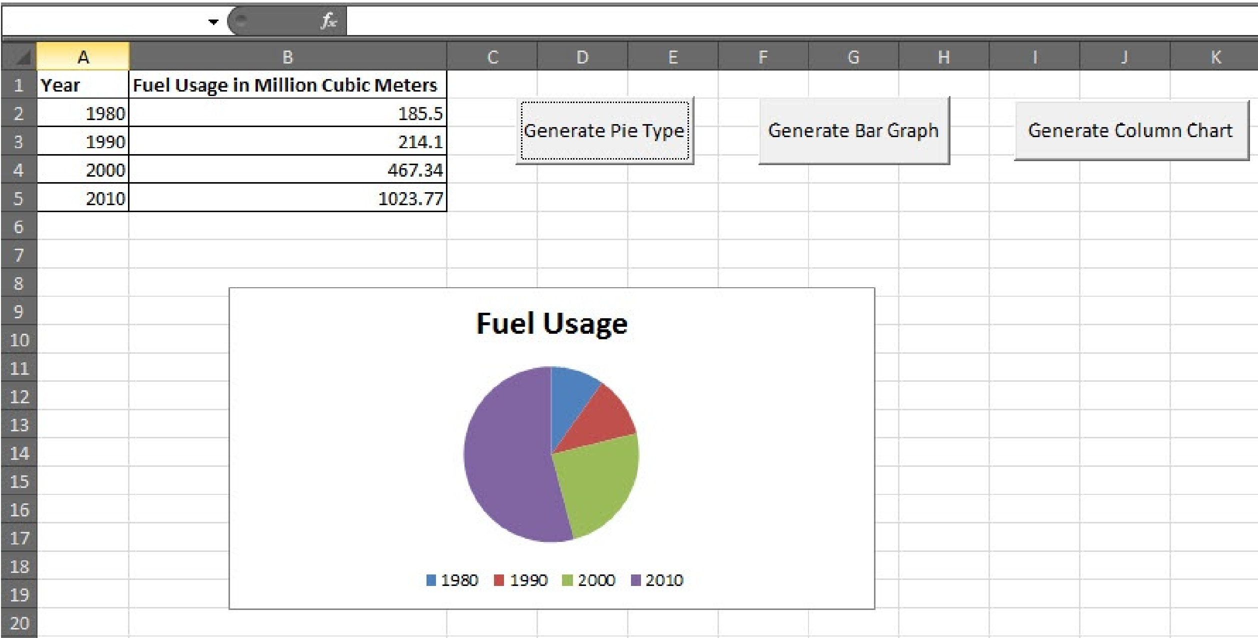
第**3**步：现在让我们建立一个宏来生成这些类型的每一个图表

```
' Procedure to Generate Pie Chart
Private Sub fn_generate_pie_graph_Click()
    Dim cht As ChartObject
    For Each cht In Worksheets(1).ChartObjects
        cht.Chart.Type = xlPie
    Next cht
End Sub

' Procedure to Generate Bar Graph
Private Sub fn_Generate_Bar_Graph_Click()
    Dim cht As ChartObject
    For Each cht In Worksheets(1).ChartObjects
        cht.Chart.Type = xlBar
    Next cht
End Sub

' Procedure to Generate Column Graph
Private Sub fn_generate_column_graph_Click()
    Dim cht As ChartObject
    For Each cht In Worksheets(1).ChartObjects
        cht.Chart.Type = xlColumn
    Next cht
End Sub
```

步骤**4**：在点击相应的按钮，创建的图表。在下面的输出，我们点击生成饼图按钮。



Year
Fuel Usage in Million Cubic Meters
1980
185.5
1990
214.1
2000
467.34
2010
1023.77
Generate Pie Type
Generate Bar Graph
Generate Column Chart
Fuel Usage
1980
1990
2000
2010

W3School VBS教程

来源：www.w3school.com.cn 整理：飞龙 日期：2014.9.30

VBScript 教程

VBScript 简介

学习之前，您需要具备的基础知识：

在继续学习之前，您应该对下面的知识有基本的了解：

- WWW, HTML 以及网站建设的基础知识

如果您希望首先学习以上的项目，请访问我们的首页。

什么是VBScript?

- VBScript 是一种脚本语言
- 脚本语言是一种轻量级的编程语言
- VBScript 是微软的编程语言 Visual Basic 的轻量级的版本

它如何工作？

当VBScript 被插入一个 HTML 文档后，因特网浏览器会读取这个文档，并对 VBScript 进行解释。VBScript 可能会立即执行，也可能在之后的事件发生时执行。

VBScript How To ...

实例

使用 VBScript 写文本

如何在页面上写文本。

```
<html>
<body>

<script type="text/vbscript">
document.write("Hello from VBScript!")
</script>

</body>
</html>
```

使用 HTML 标签格式化文本

如何协同使用 HTML 标签和 VBScript。

```
<html>
<body>

<script type="text/vbscript">
document.write("<h1>Hello World!</h1>")
document.write("<h2>Hello World!</h2>")
</script>

</body>
</html>
```

如何在 HTML 文档中放置 VBScript

```
<html>
<head>
</head>
<body>

<script type="text/vbscript">
document.write("Hello from VBScript!")
</script>

</body>
</html>
```

上面的代码会生成以下输出：

```
Hello from VBScript!
```

如需在 HTML 文档中插入脚本，请使用 <script> 标签。使用 type 属性来定义脚本语言。

```
<script type="text/vbscript">
```

然后输入 VBScript：在页面上写文本的命令是 document.write：

```
document.write("Hello from VBScript!")
```

脚本在此结束：

```
</script>
```

如何应对老式的浏览器

不支持脚本的老式浏览器会把脚本作为网页的内容显示出来。为了避免出现这样的情况，我们可以使用 HTML 的注释标签：

```
<script type="text/vbscript">
<!--
  在此输入语句
-->
</script>
```

VBScript Where To ...

实例

head 部分中的函数

可以把脚本放置在 head 部分。经常，我们会把所有的函数放置于 head 部分。这么做的目的是为了确保函数在调用前已经被载入。

```
<html>
<head>
<script type="text/vbscript">
alert("Hello")
</script>
</head>

<body>
<p>
通常，我们在 head 部分放置函数。理由是，可以确保函数在调用前已经加载。
</p>

</body>
</html>
```

body 部分中的脚本

本例会执行被放置于 body 部分的一段脚本。在 body 中的脚本会在页面载入时执行。

```
<html>
<body>

<script type="text/vbscript">
document.write("在页面加载时，会执行 body 部分的脚本。")
</script>

</body>
</html>
```

在何处放置 VBScript

当页面载入浏览器时，页面中的脚本会立即被执行。我们并不希望这种情况发生。有时我们希望当页面载入时执行脚本，而有时我们则希望当用户触发某个事件时执行这些脚本。

在 head 部分的脚本：当脚本被调用时，它们会被执行，或者某个事件被触发时，脚本也有可能会执行。当我们把脚本放置于 head 部分时，就可以确保在用户使用之前它们已经被载入了：

```
<html>
<head>
<script type="text/vbscript">
  some statements
</script>
</head>
```

在 body 部分的脚本：当页面的 body 部分被载入时，脚本就会被执行。当我们把脚本放置于 body 部分，它会生成页面的内容：

```
<html>
<head>
</head>
<body>
<script type="text/vbscript">
  some statements
</script>
</body>
```

位于 body 和 head 部分的脚本：您可以在文档中放置如何数量的脚本，因此您可以同时在 body 和 head 部分放置脚本：

```
<html>
<head>
<script type="text/vbscript">
  some statements
</script>
</head>
<body>
<script type="text/vbscript">
  some statements
</script>
</body>
```

VBScript 变量

实例

创建变量

变量用于存储信息。本例演示如何创建一个变量，并为它赋值。

```
<html>
<body>

<script type="text/vbscript">
dim name
name="John Adams"
document.write(name)
</script>

</body>
</html>
```

在一段文本中插入变量值

本例为您演示如何在一段文本中插入变量值。

```
<html>
<body>

<script type="text/vbscript">
dim name
name="John Adams"
document.write("My name is: " & name)
</script>

</body>
</html>
```

创建数组

数组用来存储一系列相关的数据项。本例演示如何创建一个存储名字的数组。（我们使用 "for loop" 来演示如何输出名字。）

```
<html>
<body>

<script type="text/vbscript">

dim fname(5)
fname(0) = "George"
fname(1) = "John"
fname(2) = "Thomas"
fname(3) = "James"
fname(4) = "Adrew"
fname(5) = "Martin"

for i=0 to 5
  document.write(fname(i) & "<br />")
next

</script>

</body>
</html>
```

什么是变量？

变量是可存储信息的“容器”。在脚本中，变量的值是可以改变的。您可以通过引用某个变量的名称，来查看或修改它的值。在 VBScript 中，所有的变量都与类型相关，可存储不同类型的数据。

变量名称的规则：

- 必须以字母开头
- 不能包含点号 (.)
- 不能超过 255 个字符

变量声明

您可以使用 Dim、Public 或 Private 语句来声明变量，比如这样：

```
dim name
name=some value
```

现在，你创建了一个变量。变量名是 "name"。

您也可以通过使用其名称来创建变量。比如这样：

```
name=some value
```

这样，您同样创建了一个名为 "name" 的变量。

不过，后面这样的做法不是一种好习惯，这是因为您可能会在脚本中拼错变量名，那样可能会在脚本运行时引起奇怪的结果。比如，当您把 "name" 变量错拼为 "nime" 时，脚本会自动创建一个名为 "nime" 的变量。为了防止脚本这样做，您可以使用 Option Explicit 语句。如果您使用这个语句，就必须使用 dim、public 或 private 语句来声明所有的变量。把 Option Explicit 语句放置于脚本的顶端，这像这样：

```
option explicit
dim name
name=some value
```

为变量赋值

您可以像这样为某个变量赋值：

```
name="George"
i=300
```

变量名在表达式的左侧，需要赋的值在表达式的右侧。现在，变量 "name" 的值是 "George"。

变量的生存期

变量的生存期指的是它可以存在的时长。

当您在一个子程序中声明变量后，变量只能在此程序内进行访问。当退出此程序时，变量也会失效。这样的变量称为本地变量。您可以在不同的子程序中使用名称相同的本地变量，因为每个变量只能在声明它的程序内得到识别。

如果您在子程序以外声明了一个变量，在您的页面上的所有子程序都可以访问访问它。这类变量的生存期始于它们被声明，止于页面被关闭。

数组变量

有时，您需要向一个单一的变量赋于多个值。那么您可以创建一个可包含一系列值的变量。这种变量被称为数组。数组变量的声明使用变量名后跟一个括号()。在下面的例子中，创建了一个包含三个元素的数组：

```
dim names(2)
```

括号中的数字是 2。数组的下标以 0 开始，因为此数组包含三个元素。这是容量固定的数组。您可以为数组的每个元素分配数据：

```
names(0)="George"
names(1)="John"
names(2)="Thomas"
```

同样地，通过使用特定数组元素的下标号，我们也可以取回任何元素的值。比如：

```
father=names(0)
```

您可以在一个数组中使用多达 60 个维数。声明多维数组的方法是在括号中用逗号来分隔数字。比如，我们声明了一个包含 5 行 7 列的 2 维数组：

```
dim table(4, 6)
```

VBScript 程序

实例

子程序

这个子程序不会返回值。

```
<html>

<head>
<script type="text/vbscript">
sub mySub()
  msgbox("这是一段子程序。")
end sub
</script>
</head>

<body>
<script type="text/vbscript">
call mySub()
</script>
<p>子程序不返回结果。</p>
</body>
</html>
```

函数程序

假如你希望返回某个值时，可以使用函数程序。

```
<html>

<head>
<script type="text/vbscript">
function myFunction()
 myFunction = "蓝色"
end function
</script>
</head>

<body>
<script type="text/vbscript">
document.write("我喜欢的颜色是：" & myFunction())
</script>
<p>函数程序可返回结果。</p>
</body>

</html>
```

VBScript 程序

我们可使用两种程序：子程序和函数程序。

子程序：

- 是一系列的语句，被封装在 Sub 和 End Sub 语句内。
- 可执行某些操作，但不会返回值。
- 可带有通过程序调用来向子程序传递参数。
- 如果没有，必须带有空的圆括号

```
Sub mysub()
 some statements
End Sub
```

```
Sub mysub(argument1,argument2)
 some statements
End Sub
```

函数程序：

- 是一系列的语句，被封装在 Function 和 End Function 语句内。
- 可执行某些操作并返回值。
- 可带有通过程序调用来向其传递参数。
- 如果没有，必须带有空的圆括号
- 通过向函数程序名赋值的方式，可使其返回值。

```
Function myfunction()
 some statements
 myfunction=some value
End Function
```

```
Function myfunction(argument1,argument2)
 some statements
 myfunction=some value
End Function
```

调用子程序或函数程序

可以这样调用某个函数：

```
name = findname()
```

此函数名为 "findname"，函数会返回一个值，这个值会存储于变量 "name" 中。

或者可以这样做：

```
msgbox "Your name is " & findname()
```

我们通过调用了名为 "findname" 的函数，这个函数返回的值会显示在消息框中。

可以这样调用子程序：

```
Call MyProc(argument)
```

或者，也可以省略 Call 语句：

```
MyProc argument
```

VBScript 条件语句

实例

If...then..else 语句

本例演示如何编写 if...then..else 语句。

```
<html>

<head>
<script type="text/vbscript">
function greeting()
i=hour(time)
if i < 10 then
 document.write("Good morning!")
else
 document.write("Have a nice day!")
end if
end function
</script>
</head>

<body onload="greeting()">
</body>

</html>
```

If...then..elseif 语句

本例演示如何编写 if...then...elseif... 语句。

```
<html>

<head>
<script type="text/vbscript">
function greeting()
i=hour(time)
If i = 10 then
    document.write("Just started...!")
elseif i = 11 then
    document.write("Hungry!")
elseif i = 12 then
    document.write("Ah, lunch-time!")
elseif i = 16 then
    document.write("Time to go home!")
else
    document.write("Unknown")
end if
end function
</script>
</head>

<body onload="greeting()">
</body>

</html>
```

Select case 语句

本例演示如何编写 select case 语句。

```
<html>

<body>
<script type="text/vbscript">
d=weekday(date)

select case d
  case 1
    document.write("Sleepy Sunday")
  case 2
    document.write("Monday again!")
  case 3
    document.write("Just Tuesday!")
  case 4
    document.write("Wednesday!")
  case 5
    document.write("Thursday...")
  case 6
    document.write("Finally Friday!")
  case else
    document.write("Super Saturday!!!!")
end select
</script>

<p>本例演示 "select case" 语句。<br />
您会得到基于日期的不同问候。<br />
请注意，Sunday=1, Monday=2, Tuesday=3, 以此类推。</p>

</body>
</html>
```

条件语句

经常地，当我们编写代码时，我们需要根据不同的判断执行不同操作。我们可以使用条件语句完成这个工作。

在 **VBScript** 我们可以使用三种条件语句：

if 语句

假如你希望在条件为 true 时执行一系列的代码，可以使用这个语句。

if...then...else 语句

假如你希望执行两套代码其中之一，可以使用这个语句。

if...then...elseif 语句

假如你希望选择多套代码之一来执行，可以使用这个语句。

select case 语句

假如你希望选择多套代码之一来执行，可以使用这个语句。

If....Then.....Else

在下面的情况中，您可以使用 If...Then...Else 语句：

- 在条件为 true 时，执行某段代码
- 选择两段代码之一来执行时

如果需要在条件为 true 时只执行一行语句，可以把代码写为一行：

```
if i=10 Then msgbox "Hello"
```

在上面的代码中，没有 .else.. 语句。我们仅仅让代码在条件为 true 时执行一项操作（当 i 为 10 时）。

假如我们需要在条件为 true 时执行不止一条语句，那么就必须在一行写一条语句，然后使用关键词 "End If" 来结束这个语句：

```
if i=10 Then
   msgbox "Hello"
   i = i+1
end If
```

在上面的代码中，同样没有 .else.. 语句。我们仅仅让代码在条件为 true 时执行了多项操作。

假如我们希望在条件为 true 时执行某条语句，并当条件不为 true 时执行另一条语句，就必须添加关键词 "Else"：

```
if i=10 then
   msgbox "Hello"
else
   msgbox "Goodbye"
end If
```

当条件为 true 时会执行第一段代码，当条件不成立时执行第二段代码（当 i 不等于 10 时）。

If....Then.....Elseif

假如你希望选择多套代码之一来执行，可以使用if...then...elseif语句：

```
if payment="Cash" then
   msgbox "You are going to pay cash!"
 elseif payment="Visa" then
   msgbox "You are going to pay with visa."
 elseif payment="AmEx" then
   msgbox "You are going to pay with American Express."
 else
   msgbox "Unknown method of payment."
end If
```

Select Case

假如你希望选择多套代码之一来执行，可以使用 SELECT 语句：

```
select case payment
 case "Cash"
   msgbox "You are going to pay cash"
 case "Visa"
   msgbox "You are going to pay with visa"
 case "AmEx"
   msgbox "You are going to pay with American Express"
 case Else
   msgbox "Unknown method of payment"
end select
```

以上代码的工作原理：首先，我们需要一个简单的表达式（常常是一个变量），并且这个表达式会被做一次求值运算。然后，表达式的值会与每个 case 中的值作比较，如果匹配，被匹配的 case 所对应的代码会被执行。

VBScript 循环语句

实例

For..next 循环

本例演示如何编写一个简单的 For....Next 循环。

```
<html>
<body>

<script type="text/vbscript">
for i = 0 to 5
 document.write("数字是：" & i & "<br />")
next
</script>

</body>
</html>
```

循环输出HTML标题

本例演示如何循环生成 6 个 HTML 标题。

```
<html>
<body>

<script type="text/vbscript">
for i=1 to 6
 document.write("<h" & i & ">这是标题 " & i & "</h" & i & ">")
next
</script>

</body>
</html>
```

For..each 循环

本例演示如何编写一个简单的 For.....Each 循环。

```
<html>
<body>

<script type="text/vbscript">
dim names(2)
names(0) = "George"
names(1) = "John"
names(2) = "Thomas"

for each x in names
  document.write(x & "<br />")
next
</script>

</body>
</html>
```

Do...While 循环

本例演示如何编写简单的 Do...While 循环。

```
<html>
<body>

<script type="text/vbscript">
i=0
do while i < 10
  document.write(i & "<br />")
  i=i+1
loop
</script>

</body>
</html>
```

Looping 语句

经常地，当编写代码时，我们希望将一段代码执行若干次。我们可以在代码中使用循环语句来完成这项工作。

在 VBScript 中，我们可以使用四种循环语句：

For...Next 语句

运行一段语句指定的次数

For Each...Next 语句

针对集合中的每个项目或者数组中的每个元素来运行某段语句。

Do...Loop 语句

运行循环，当条件为 true 或者直到条件为 true 时。

While...Wend 语句

不要使用这个语句 - 请使用 Do...Loop 语句代替它。

For...Next 循环

如果您已经确定需要重复执行代码的次数，那么您可以使用 For...Next 语句来运行这段代码。

我们可以使用一个计数器变量，这个变量会随着每次循环递增或递减，例如这样：

```
For i=1 to 10
  some code
Next
```

For 语句规定计数变量以及它的开始值和结束值。

Next 语句会以 1 作为步进值来递增变量i。

Step 关键词

通过使用 Step 关键词，我们可以规定计数变量递增或递减的步进值。

在下面的例子中，计数变量i每次循环的递增步进值为 2。

```
For i=2 To 10 Step 2
  some code
Next
```

如果要递减计数变量，就必须负的步进值。并且需要规定小于开始值的结束值。

在下面的例子中，计数变量i每次循环的递减步进值为 2。

```
For i=10 To 2 Step -2
  some code
Next
```

退出 For...Next

如需退出 For...Next 语句，可以使用 Exit 关键词。

VBScript 参考

VBScript Date/Time 函数

函数	描述
CDate	把一个有效的日期或时间表达式转换为日期类型。
Date	返回当前的系统日期。
DateAdd	返回已添加指定时间间隔的日期。
DateDiff	返回两个日期之间的时间间隔数。
DatePart	返回给定日期的指定部分。
DateSerial	返回日期的指定年、月、日
DateValue	返回日期
Day	返回代表一月中一天的数字 （介于并包括1至31之间）
FormatDateTime	返回以日期或时间格式化的表达式。
Hour	返回可代表一天中的小时的数字 （介于并包括0至23之间）
IsDate	返回可指示计算表达式能否转换为日期的布尔值。
Minute	返回一个数字，代表小时的分钟 （介于并包括0至59）
Month	返回一个数字，代表年的月份 （介于并包括1至12之间）。
MonthName	返回指定月份的名称。
Now	返回当前的系统日期和时间。
Second	返回一个数字，代表分钟的秒 （介于并包括0至59之间）
Time	返回当前的系统时间。
Timer	返回自 12:00 AM 以来的秒数。
TimeSerial	返回特定小时、分钟和秒的时间。
TimeValue	返回时间。
Weekday	返回一个数字，代表星期的一天（介于并包括1至7）
WeekdayName	返回星期中指定的一天的星期名。
Year	返回一个代表年份的数字。

VBScript CDate 函数

VBScript 函数参考手册

定义和用法

CDate 函数可把一个合法的日期和时间表达式转换为 Date 类型，并返回结果。

提示：请使用 IsDate 函数来判断 date 是否可被转换为日期或时间。

注释：IsDate 函数使用本地设置来检测字符串是否可被转换为日期。

语法

```
CDate(date)
```

参数	描述
date	必需的。任何有效的日期表达式。（比如 Date() 或者 Now()）

实例

例子 1

```
d="April 22, 2001"
if IsDate(d) then
  document.write(CDate(d))
end if
```

输出：

```
2/22/01
```

例子 2

```
d=#2/22/01#
if IsDate(d) then
  document.write(CDate(d))
end if
```

输出：

```
2/22/01
```

例子 3

```
d="3:18:40 AM"
if IsDate(d) then
  document.write(CDate(d))
end if
```

输出：

```
3:18:40 AM
```

VBScript 函数参考手册

VBScript Date 函数

VBScript 函数参考手册

定义和用法

Date 函数可返回当前的系统日期。

语法

```
Date
```

提示和注释

重要事项：

如果同时读取 Date、Time 以及 Now，那么 Now = Date + Time，但是实际上，我们不可能同时调用这三个函数，因为执行完一个函数之后，才能执行另一个函数，所以如果您在程序中必需同时取得当时的日期和时间，必需调用 Now，再利用 DateVale 及 TimeValue 分别取出日期和时间。

实例：取得某一时间点的日期和时间：

```
N = Now '这个时间点的日期和时间
D = Datevalue(N) '同一时间点的日期部分
T = TimeValue(N) '同一时间点的时间部分
D2 = Date '时间点1的日期
T2 = Time '时间点2的时间
```

问题思考

连续执行 Response.write Now 及 Response.Write Date + Time，则可能出现的最大误差值有多大？假设：

```
时间点1取得的    Now = #7/1/95 23:59:59#
时间点2取得的    Date = #7/1/95#
```

而如果“时间点3”刚好跨过一日，所以 Time = #0:00:00，于是 Now 与 Date+Time 的差距便成了23:59:59。

实例

例子 1

```
document.write("The current system date is: ")
document.write(Date)
```

输出：

```
The current system date is: 1/14/2002
```

TIY

Date

如何使用 Date 函数来显示当前日期。

VBScript 函数参考手册

VBScript DateAdd 函数

VBScript 函数参考手册

定义和用法

DateAdd 函数可返回已添加指定时间间隔的日期。

语法

```
DateAdd(interval,number,date)
```

参数	描述
interval	必需的。需要增加的时间间隔。可采用下面的值： `yyyy` - 年 `q` - 季度 `m` - 月 `y` - 当年的第几天 `d` - 日 `w` - 当周的第几天 `ww` - 周 `h` - 小时 `n` - 分钟 `s` - 秒
number	必需的。需要添加的时间间隔的数目。可对未来的日期使用正值，对过去的日期使用负值。
date	必需的。代表被添加的时间间隔的日期的变量或文字。

实例

例子 1

给 January 31, 2000 增加一个月：

```
document.write(DateAdd("m",1,"31-Jan-00"))
```

输出：

```
2/29/2000
```

例子 1

给 January 31, 2001 增加一个月：

```
document.write(DateAdd("m",1,"31-Jan-01"))
```

输出：

```
2/28/2001
```

例子 1

从 January 31, 2001 减去一个月：

```
document.write(DateAdd("m",-1,"31-Jan-01"))
```

输出：

```
12/31/2000
```

TIY

DateAdd

如何使用 DateAdd 函数为日期增加一个月。

DateAdd

如何使用 DateAdd 函数从日期减去一个月。

VBScript 函数参考手册

VBScript DateDiff 函数

VBScript 函数参考手册

定义和用法

DateDiff 函数可返回两个日期之间的时间间隔数。

DateDiff 函数用于计算两日期时间的差值，计算方法是 date2 - date1。

若比较年份，则不管月份以下的数值，若比较月份，则不管天数以下的数值...... 以此类推。

注释：firstdayofweek 参数会对使用“w”和“ww”间隔符号的计算产生影响。

语法

```
DateDiff(interval,date1,date2[,firstdayofweek[,firstweekofyear]])
```

参数	描述
interval	必需的。计算 date1 和 date2 之间的时间间隔的单位。可采用下面的值： `yyyy` - 年 `q` - 季度 `m` - 月 `y` - 当年的第几天 `d` - 日 `w` - 当周的第几天 `ww` - 周 `h` - 小时 `n` - 分钟 `s` - 秒
date1,date2	必需的。日期表达式。在计算中需要使用的两个日期。
firstdayofweek	可选的。规定一周的日数，即当周的第几天。可采用下面的值： 0 = `vbUseSystemDayOfWeek` - 使用区域语言支持 (NLS) API 设置。 1 = `vbSunday` - 星期日 (默认) 2 = `vbMonday` - 星期一 3 = `vbTuesday` - 星期二 4 = `vbWednesday` - 星期三 5 = `vbThursday` - 星期四 6 = `vbFriday` - 星期五 7 = `vbSaturday` - 星期六
firstweekofyear	可选的。规定一年中的第一周。可采用下面的值： 0 = `vbUseSystem` - 使用区域语言支持 (NLS) API 设置。 1 = `vbFirstJan1` - 由 1 月 1 日所在的星期开始（默认）。 2 = `vbFirstFourDays` - 由在新年中至少有四天的第一周开始。 3 = `vbFirstFullWeek` - 由在新的一年中第一个完整的周开始。

实例

例子 1

```
document.write(Date & "<br />")
document.write(DateDiff("m",Date,"12/31/2002") & "<br />")
document.write(DateDiff("d",Date,"12/31/2002") & "<br />")
document.write(DateDiff("n",Date,"12/31/2002"))
```

输出：

```
1/14/2002
11
351
505440
```

例子 2

请注意在下面的代码中，date1>date2：

```
document.write(Date & "<br />")
document.write(DateDiff("d","12/31/2002",Date))
```

输出：

```
1/14/2002
-351
```

例子 3

```
'How many weeks (start on Monday),
'are left between the current date and 10/10/2002
document.write(Date & "<br />")
document.write(DateDiff("w",Date,"10/10/2002",vbMonday))
```

输出：

```
1/14/2002
38
```

VBScript 函数参考手册

VBScript DatePart 函数

VBScript 函数参考手册

定义和用法

DatePart 函数可返回给定日期的指定部分。

注释：firstdayofweek 参数会对使用“w”和“ww”间隔符号的计算产生影响。

语法

```
DatePart(interval,date[,firstdayofweek[,firstweekofyear]])
```

参数	描述
interval	必需的。计算 date1 和 date2 之间的时间间隔的单位。可采用下面的值： `yyyy` - 年 `q` - 季度 `m` - 月 `y` - 当年的第几天 `d` - 日 `w` - 当周的第几天 `ww` - 周 `h` - 小时 `n` - 分钟 `s` - 秒
date	必需的。需计算的日期表达式。
firstdayofweek	可选的。规定一周的日数，即当周的第几天。可采用下面的值： 0 = `vbUseSystemDayOfWeek` - 使用区域语言支持 (NLS) API 设置。 1 = `vbSunday` - 星期日 (默认) 2 = `vbMonday` - 星期一 3 = `vbTuesday` - 星期二 4 = `vbWednesday` - 星期三 5 = `vbThursday` - 星期四 6 = `vbFriday` - 星期五 7 = `vbSaturday` - 星期六
firstweekofyear	可选的。规定一年中的第一周。可采用下面的值： 0 = `vbUseSystem` - 使用区域语言支持 (NLS) API 设置。 1 = `vbFirstJan1` - 由 1 月 1 日所在的星期开始（默认）。 2 = `vbFirstFourDays` - 由在新年中至少有四天的第一周开始。 3 = `vbFirstFullWeek` - 由在新的一年中第一个完整的周开始。

实例

例子 1

```
d = #2/10/96 16:45:30#
document.write(DatePart("yyyy",d)) '输出：1996
document.write(DatePart("m",d)) '输出：2
document.write(DatePart("d",d)) '输出：10
document.write(DatePart("h",d)) '输出：16
document.write(DatePart("n",d)) '输出：45
document.write(DatePart("s",d)) '输出：30
document.write(DatePart("q",d)) '输出：1，2月是第1季
document.write(DatePart("y",d)) '输出：41，2月10日是1996年的第41日。
document.write(DatePart("ww",d)) '输出：6，2月10日是1996年的第6周。
document.write(DatePart("w",d)) '输出：7，2月10日在在1996年是第6周的第7日（星期六）。
```

VBScript 函数参考手册

VBScript DateSerial 函数

VBScript 函数参考手册

定义和用法

DateSerial 函数可返回指定的年、月、日的子类型 Date 的 Variant 。

也就是说，**DateSerial** 函数可以把年、月、日合并为日期。

语法

```
DateSerial(year,month,day)
```

参数	描述
year	必需的。介于100到9999的数字，或数值表达式。介于 0 到 99 的值被视为 1900–1999。对于所有其他的 year 参数，请使用完整的4位年份。
month	必需的。任何数值表达式。若大于12，则日期从12月起向后推算mouth-12个月，若小于1，则日期从1月起向前推算1-month个月。
day	必需的。任何数值表达式。若大于当月的日数，则日期从当月日数起，向后推算day-当月日数；若小于1，则日期从1日起向前推算1-day日。

实例

例子 1

```
document.write(DateSerial(1996,2,3)) '普通的调用方法
```

输出：

```
1996/2/3
```

例子 2

```
document.write(DateSerial(95,13,10)) '13月=1年+1月
```

输出：

```
1996/01/10
```

例子 3

```
document.write(DateSerial(96,-1,10)) '-1月要从1月起向前推算1-（-1）=2个月
```

输出：

```
1995/11/10
```

例子 4

```
document.write(DateSerial(95,2,30)) '95年2月有28日，所以30日=1月+2日
```

输出：

```
1995/03/02
```

例子 5

```
document.write(DateSerial(95,2,-2)) '-2日要从1日起向前推算1-（-2）=3日
```

输出：

```
1995/01/29
```

例子

```
document.write(DateSerial(1990-20,9-2,1-1))
'1990-20=1970年，9-2=7月，1-1=0日，0日要从1日起向前推算1-0=1日。
```

输出：

```
1970/6/30
```

VBScript 函数参考手册

VBScript DateValue 函数

VBScript 函数参考手册

定义和用法

DateValue 函数可返回一个日期类型。

也就是说，**DateValue** 函数可从表达式中取回日期。

注释：如果日期中的年份部分不省略，那么函数会使用来自计算机系统日期的当前年份。

注释：如果日期参数含有时间信息，那么时间信息不会被返回。如果日期中含有无效的时间信息，那么会发生 run-time 错误。

注释：如果参数中不含日期，那么 DateValue 函数将返回0，若输出则是12:00:00 AM，即0。

语法

```
DateValue(date)
```

参数	描述
date	必需的。一个介于100年1月1日到9999年12月31日的日期，或者任何可表示日期、时间或日期时间兼而有之的表达式。

实例

例子 1

```
document.write(DateValue("31-Jan-02"))
```

输出：

```
1/31/2002
```

例子 2

```
document.write(DateValue("31-Jan")) '假设当年是2002年, 则输出的是系统日期的年份
```

输出：

```
1/31/2002
```

例子 3

```
document.write(DateValue("31-Jan-02 2:39:49 AM"))
'如果参数同时包含日期和时间, 则只输出日期。
```

输出：

```
1/31/2002
```

例子 4

```
document.write(DateValue("2:39:49 AM")) '输出 12:00:00 AM , 相当于0。
```

输出：

```
12:00:00 AM
```

VBScript 函数参考手册

VBScript Day 函数

VBScript 函数参考手册

定义和用法

Day 函数可返回介于 1 到 31 之间的一个代表月的天数的数字。

语法

```
Day(date)
```

参数	描述
date	必需的。可表示一个日期的表达式。

实例

例子 1

```
document.write(Date & "<br />")
document.write(Day(Date))
```

输出：

```
1/14/2002
14
```

VBScript 函数参考手册

VBScript FormatDateTime 函数

VBScript 函数参考手册

定义和用法

FormatDateTime 函数可格式化并返回一个合法的日期或时间表达式。

语法

```
FormatDateTime(date,format)
```

参数	描述
date	必需的。任何合法的日期表达式。(比如 Date() 或 Now())
format	可选的。规定所使用的日期/时间格式的格式值。

format 参数：

常数	值	描述
vbGeneralDate	0	显示日期和/或时间。如果有日期部分，则将该部分显示为短日期格式。如果有时间部分，则将该部分显示为长时间格式。如果都存在，则显示所有部分。
vbLongDate	1	使用计算机区域设置中指定的长日期格式显示日期。
vbShortDate	2	使用计算机区域设置中指定的短日期格式显示日期。
vbLongTime	3	使用此格式显示时间：hh:mm:ss PM/AM
vbShortTime	4	使用 24 小时格式 (hh:mm) 显示时间。

实例

例子 1

```
D = #2001/2/22#
document.write(FormatDateTime(D))
```

输出：

```
2001-2-22
```

例子 2

```
D = #2001/2/22#
document.write(FormatDateTime(D,1))
```

输出：

```
2001年2月22日
```

例子 3

```
D = #2001/2/22#
document.write(FormatDateTime(D,2))
```

输出：

```
2001-2-22
```

例子 4

```
D = #2001/2/22#
document.write(FormatDateTime(D,3))
```

输出：

```
2001-2-22
```

VBScript 函数参考手册

VBScript Hour 函数

VBScript 函数参考手册

定义和用法

Hour 函数可返回介于 0 到 23 之间的代表天的小时数的数字。

语法

```
Hour(time)
```

参数	描述
time	必需的。任何可表示时间的表达式。

实例

例子 1

```
T = #1/15/2002 10:07:47 AM#
document.write(Hour(T))
```

输出：

```
10
```

例子 2

```
T = #10:07:47 AM#
document.write(Hour(T))
```

输出：

```
10
```

VBScript 函数参考手册

VBScript IsDate 函数

VBScript 函数参考手册

定义和用法

IsDate 函数可一个布尔值，指示经计算的表达式是否可被转换为日期。如果表达式是日期，或可被转换为日期，则返回 True 。否则，返回 False 。

注释：IsDate 函数使用本地设置来检测字符串是否可以转换为日期。

语法

```
IsDate(expression)
```

参数	描述
expression	必需的。要计算的表达式。

实例

例子 1

```
document.write(IsDate("April 22, 1947"))
```

输出：

```
True
```

例子 2

```
document.write(IsDate(#11/11/01#))
```

输出：

```
True
```

例子 3

```
document.write(IsDate("#11/11/01#"))
```

输出：

```
False
```

例子 4

```
document.write(IsDate("Hello World!"))
```

输出：

```
False
```

VBScript 函数参考手册

VBScript Minute 函数

VBScript 函数参考手册

定义和用法

Minute 函数可返回表示小时的分钟数的数字，介于0到59。

语法

```
Minute(time)
```

参数	描述
time	必需的。表示时间的表达式。

实例

例子 1

```
D = #1/15/2002 10:34:39 AM#
document.write(Minute(D))
```

输出：

```
34
```

例子 2

```
T = #10:34:39 AM#
document.write(Minute(T))
```

输出：

```
34
```

VBScript 函数参考手册

VBScript Month 函数

VBScript 函数参考手册

定义和用法

Month 函数可返回表示年的月份的数字，介于 1 到 12。

语法

```
Month(date)
```

参数	描述
date	必需的。任何可表示日期的表达式。

实例

例子 1

```
D = #1/15/2002#
document.write(Month(D))
```

输出：

```
1
```

VBScript 函数参考手册

VBScript MonthName 函数

VBScript 函数参考手册

定义和用法

MonthName 函数可返回指定的月份的名称。

语法

```
MonthName(month[,abbreviate])
```

参数	描述
month	必需的。规定月的数字。（比如一月是1，二月是2，依此类推。）
abbreviate	可选的。一个布尔值，指示是否缩写月份名称。默认是 False。

实例

例子 1

```
document.write(MonthName(8))
```

输出：

```
八月
```

例子 2

```
document.write(MonthName(8,true))
```

输出：

```
Aug
```

注释：在中文系统中，仍然输出为“八月”。

VBScript 函数参考手册

VBScript Now 函数

VBScript 函数参考手册

定义和用法

Now 函数可根据计算机系统的日期和时间设置返回当前的日期和时间。

语法

```
Now
```

提示和注释

重要事项：

如果同时读取 Date、Time 以及 Now，那么 Now = Date + Time，但是实际上，我们不可能同时调用这三个函数，因为执行完一个函数之后，才能执行另一个函数，所以如果您在程序中必需同时取得当时的日期和时间，必需调用 Now，再利用 DateVale 及 TimeValue 分别取出日期和时间。

实例：取得某一时间点的日期和时间：

```
N = Now '这个时间点的日期和时间
D = Datevalue(N) '同一时间点的日期部分
T = TimeValue(N) '同一时间点的时间部分
D2 = Date '时间点1的日期
T2 = Time '时间点2的时间
```

问题思考

连续执行 Response.write Now 及 Response.Write Date + Time，则可能出现的最大误差值有多大？假设：

```
时间点1取得的    Now = #7/1/95 23:59:59#
时间点2取得的    Date = #7/1/95#
```

而如果“时间点3”刚好跨过一日，所以 Time = #0:00:00，于是 Now 与 Date+Time 的差距便成了23:59:59。

实例

例子 1

```
document.write(Now)
```

输出：

```
2007-10-1 14:10:06
```

注释：输出的结果可能由于不同的计算机设置而略有差异。

VBScript 函数参考手册

VBScript Second 函数

VBScript 函数参考手册

定义和用法

Second 函数可返回表示分钟的秒数的数字，介于 0 到59 之间。

语法

```
Second(time)
```

参数	描述
time	必需的。任何可表示时间的表达式。

实例

例子 1

```
D = #1/15/2002 10:55:51 AM#
document.write(Second(D))
```

输出：

```
51
```

例子 2

```
T = #10:55:51 AM#
document.write(Second(T))
```

输出：

```
51
```

例子 3

下面这个例子是一个可输出中文格式时间的 ASP 函数，其中使用到了 VBScript 的 Hour、Minute 以及 Second 函数。

```
<%
Sub HHMMSS(T)
H = Hour(T)
If H<12 then
  Response.Write "上午"&H&"时"
Else
  Response.Write "下午"&(H-12)&"时"
End If
Response.Write Minute(T)&"分"
Response.Write Second(T)&"秒"
End Sub
%>
```

VBScript 函数参考手册

VBScript Time 函数

VBScript 函数参考手册

定义和用法

Time 函数可返回当前的系统时间。

语法

```
Time
```

提示和注释

重要事项：

如果同时读取 Date、Time 以及 Now，那么 Now = Date + Time，但是实际上，我们不可能同时调用这三个函数，因为执行完一个函数之后，才能执行另一个函数，所以如果您在程序中必需同时取得当时的日期和时间，必需调用 Now，再利用 DateVale 及 TimeValue 分别取出日期和时间。

实例：取得某一时间点的日期和时间：

```
N = Now '这个时间点的日期和时间
D = Datevalue(N) '同一时间点的日期部分
T = TimeValue(N) '同一时间点的时间部分
D2 = Date '时间点1的日期
T2 = Time '时间点2的时间
```

问题思考

连续执行 Response.write Now 及 Response.Write Date + Time，则可能出现的最大误差值有多大？假设：

```
时间点1取得的    Now = #7/1/95 23:59:59#
时间点2取得的    Date = #7/1/95#
```

而如果“时间点3”刚好跨过一日，所以 Time = #0:00:00，于是 Now 与 Date+Time 的差距便成了23:59:59。

实例

例子 1

```
document.write(Time)
```

输出：

```
14:34:38
```

注释：输出的结果可能由于不同的计算机设置而略有差异。

VBScript 函数参考手册

VBScript Timer 函数

VBScript 函数参考手册

定义和用法

Timer 函数可返回午夜 12 时（12:00 AM）以后已经过去的秒数。

语法

```
Timer
```

实例

例子 1

```
document.write(Timer)
```

输出：

```
52744.64
```

VBScript 函数参考手册

VBScript TimeSerial 函数

VBScript 函数参考手册

定义和用法

TimeSerial 函数可把时、分、秒合并成为时间。

注释：时分秒若超过应有的范围，其推算的原理与 DateSerial 相同。若经推算后得到的时间小于 #00:00:00#，则自动将负时间变为正时间；若经推算后得到的时间大于等于 #24:00:00#，则时间向前增加，使数据变成一个含有日期时间的数据，其中日期的起算日是 #12/30/1899#。

语法

```
TimeSerial(hour,minute,second)
```

参数	描述
hour	必需的。介于 0-23 的数字，或数值表达式。
minute	必需的。介于 0-59 的数字，或数值表达式。
second	必需的。介于 0-59 的数字，或数值表达式。

要指定一时刻，如 11:59:59，TimeSerial 的参数取值应在可接受的范围内；也就是说，小时应介于 0-23 之间，分和秒应介于 0-59 之间。但是，可以使用数值表达式为每个参数指定相对时间，这一表达式代表某时刻之前或之后的时、分或秒数。

当任何一个参数的取值超出可接受的范围时，它会正确地进位到下一个较大的时间单位中。例如，如果指定了 75 分钟，则这个时间被解释成一小时十五分钟。但是，如果任何一个参数值超出 -32768 到 32767 的范围，就会导致错误。如果使用三个参数直接指定的时间或通过表达式计算出的时间超出可接受的日期范围，也会导致错误。

实例

例子 1

```
document.write(TimeSerial(9,30,50)) '正常的调用方法
```

输出：

```
9:30:50 或 9:30:50 AM
```

例子 2

```
document.write(TimeSerial(0,9,11)) '正常的调用方法
```

输出：

```
0:09:11 或 12:09:11 AM
```

例子 3

```
document.write(TimeSerial(14+2,9-2,1-1)) '根据数值表达式的结果来输出时间
```

输出：

```
16:07:00 或 4:07:00 PM
```

例子 4

```
document.write(TimeSerial(26,30,0)) '日期从#12/30/1899#起向后增加1日
```

输出：

```
1899-12-31 2:30:00 AM
```

VBScript 函数参考手册

VBScript TimeValue 函数

VBScript 函数参考手册

定义和用法

TimeValue 函数可返回包含时间的日期子类型的变量。

TimeValue 函数可用来取出参数的时间部分。

注释：若参数不含时间，则 TimeValue 将返回0，若输出结果则是 12:00:00 AM。

语法

```
TimeValue(time)
```

参数	描述
time	必需的。介于 0:00:00 (12:00:00 A.M.) - 23:59:59 (11:59:59 P.M.) 的时间，或任何可表示此范围内时间的表达式。

实例

例子 1

```
D = #7/1/96 13:30:00#
document.write(TimeValue(D))
```

输出：

```
13:30:00 或 1:30:00 PM
```

例子 2

```
D = "7/1/96 13:30:00"
document.write(TimeValue(D)) '只要能够判断，字符串也能接受。
```

输出：

```
13:30:00 或 1:30:00 PM
```

例子 3

```
D = "7/1/96"
document.write(TimeValue(D)) '只有日期
```

输出：

```
12:00:00 AM '相当于0
```

VBScript 函数参考手册

VBScript Weekday 函数

VBScript 函数参考手册

定义和用法

Weekday 函数可返回表示一周的天数的数字，介于 1 和 7 之间。

注释：

语法

```
Weekday(date[,firstdayofweek])
```

参数	描述
date	必需的。需计算的日期表达式。
firstdayofweek	可选的。规定周的第一天。可采用下面的值： 0 = vbUseSystemDayOfWeek - 使用区域语言支持 (NLS) API 设置。 1 = vbSunday - 星期日（默认） 2 = vbMonday - 星期一 3 = vbTuesday - 星期二 4 = vbWednesday - 星期三 5 = vbThursday - 星期四 6 = vbFriday - 星期五 7 = vbSaturday - 星期六

实例

例子 1

```
D = #10/1/2007#
document.write(Weekday(D)) '2007年10月1日是星期一，即一周的第二天。
```

输出：

```
2
```

VBScript 函数参考手册

VBScript WeekdayName 函数

VBScript 函数参考手册

定义和用法

WeekdayName 函数可返回一周中指定一天的星期名。

注释：在英文 Windows 环境下，返回值是 "Sunday"、"Monday" 、 ... ...

语法

```
WeekdayName(weekday[,abbreviate[,firstdayofweek]])
```

参数	描述
weekday	必需的。一周的第几天的数字。
abbreviate	可选的。布尔值，指示是否缩写星期名。
firstdayofweek	可选的。规定周的第一天。可采用下面的值： 0 = `vbUseSystemDayOfWeek` - 使用区域语言支持 (NLS) API 设置。 1 = `vbSunday` - 星期日（默认） 2 = `vbMonday` - 星期一 3 = `vbTuesday` - 星期二 4 = `vbWednesday` - 星期三 5 = `vbThursday` - 星期四 6 = `vbFriday` - 星期五 7 = `vbSaturday` - 星期六

实例

例子 1

```
document.write(WeekdayName(3))
```

输出：

```
星期二
```

例子 2

```
D = #2007/10/1#
document.write(WeekdayName(Weekday(D)))
```

输出：

```
星期一
```

例子 3

```
D = #2007/10/1#
document.write(WeekdayName(Weekday(Date),true))
```

输出：

```
星期一 或 Mon
```

VBScript 函数参考手册

VBScript Year 函数

VBScript 函数参考手册

定义和用法

Year 函数可返回表示年份的一个数字。

语法

```
Year(date)
```

参数	描述
date	必需的。能表示日期的任何表达式。

实例

例子 1

```
D = #2007/10/1#
document.write(Year(D))
```

输出：

```
2007
```

VBScript 函数参考手册

VBScript Conversion 函数

函数	描述
Asc	把字符串中的首字母转换为 ANSI 字符代码。
CBool	把表达式转换为布尔类型。
CByte	把表达式转换为字节（Byte）类型。
CCur	把表达式转换为货币（Currency）类型。
CDate	把有效的日期和时间表达式转换为日期（Date）类型。
CDbl	把表达式转换为双精度（Double）类型。
Chr	把指定的 ANSI 字符代码转换为字符。
CInt	把表达式转换为整数（Integer）类型。
CLng	把表达式转换为长整形（Long）类型。
CSng	把表达式转换为单精度（Single）类型。
CStr	把表达式转换为子类型 String 的 variant 。
Hex	返回指定数字的十六进制值。
Oct	返回指定数字的八进制值。

VBScript Asc 函数

VBScript 函数参考手册

定义和用法

Asc 函数可把字符串中的第一个字母转换为对应的 ANSI 代码，并返回结果。

语法

```
Asc(string)
```

参数	描述
string	必需的。字符串表达式。不能为空字符串！

实例

例子 1

```
document.write(Asc("A"))
document.write(Asc("F"))
```

输出分别是：

```
65
70
```

例子 2

```
document.write(Asc("a"))
document.write(Asc("f"))
```

输出分别是：

```
97
102
```

例子 3

```
document.write(Asc("W"))
document.write(Asc("W3School.com.cn"))
```

输出分别是：

```
87
87
```

例子 4

```
document.write(Asc("2"))
document.write(Asc("#"))
```

输出分别是：

```
50
35
```

VBScript CBool 函数

VBScript 函数参考手册

定义和用法

CBool 函数可把表达式转换为布尔类型。

语法

```
CBool(expression)
```

参数	描述
expression	必需的。任何合法的表达式。非零的值返回 True，零返回 False。如果表达式不能解释为数值，则将发生 run-time 错误。

实例

例子 1

```
dim a,b
a=5
b=10
document.write(CBool(a))
document.write(CBool(b))
```

输出分别是：

```
True
True
```

VBScript 函数参考手册

VBScript CByte 函数

VBScript 函数参考手册

定义和用法

CByte 函数可把表达式转换为字节（Byte）类型。

语法

```
CByte(expression)
```

参数	描述
expression	必需的。任何合法的表达式。

实例

例子 1

```
dim a
a=134.345
document.write(CByte(a))
```

输出：

```
134
```

例子 2

```
dim a
a=14.345455
document.write(CByte(a))
```

输出：

```
14
```

VBScript 函数参考手册

VBScript CCur 函数

VBScript 函数参考手册

定义和用法

CCur 函数可把表达式转换为货币（Currency）类型。

语法

```
CCur(expression)
```

参数	描述
expression	必需的。任何合法的表达式。

实例

例子 1

```
dim a
a=134.345
document.write(CCur(a))
```

输出：

```
134.345
```

例子 2

```
dim a
a=1411111111.345455
document.write(CCur(a))    '请注意，此函数会把结果四舍五入为4位的小数。
```

输出：

```
1411111111.3455
```

VBScript 函数参考手册

VBScript CDate 函数

VBScript 函数参考手册

定义和用法

CDate 函数可把一个合法的日期和时间表达式转换为 Date 类型，并返回结果。

提示：请使用 IsDate 函数来判断 date 是否可被转换为日期或时间。

注释：IsDate 函数使用本地设置来检测字符串是否可被转换为日期。

语法

```
CDate(date)
```

参数	描述
date	必需的。任何有效的日期表达式。（比如 Date() 或者 Now()）

实例

例子 1

```
d="April 22, 2001"
if IsDate(d) then
  document.write(CDate(d))
end if
```

输出：

```
2/22/01
```

例子 2

```
d=#2/22/01#
if IsDate(d) then
  document.write(CDate(d))
end if
```

输出：

```
2/22/01
```

例子 3

```
d="3:18:40 AM"
if IsDate(d) then
  document.write(CDate(d))
end if
```

输出：

```
3:18:40 AM
```

VBScript 函数参考手册

VBScript CDbl 函数

VBScript 函数参考手册

定义和用法

CDbl 函数可把表达式转换为双精度（Double）类型。

语法

```
CDbl(expression)
```

参数	描述
expression	必需的。任何合法的表达式。

实例

例子 1

```
dim a
a=134.345
document.write(CDbl(a))
```

输出：

```
134.345
```

例子 2

```
dim a
a=14111111113353355.345455
document.write(CDbl(a))
```

输出：

```
1.41111111133534E+16
```

VBScript 函数参考手册

VBScript Chr 函数

VBScript 函数参考手册

定义和用法

Chr 函数可把指定的 ANSI 字符代码转换为字符。

注释：从0到31的数字代表不可打印的 ASCII 代码，举例，Chr(10) 会返回一个换行符。

语法

```
Chr(charcode)
```

参数	描述
charcode	必需的。标识某个字符的数字。

实例

例子 1

```
document.write(Chr(65))
document.write(Chr(97))
```

输出分别是：

```
A
a
```

例子 2

```
document.write(Chr(37))
document.write(Chr(45))
```

输出分别是：

```
%
-
```

例子 2

```
document.write(Chr(50))
document.write(Chr(35))
```

输出分别是：

```
2
#
```

VBScript 函数参考手册

VBScript CInt 函数

VBScript 函数参考手册

定义和用法

CInt 函数可把表达式转换为整数（Integer）类型。

注释：值必须是介于 -32768 与 32767 之间的数字。

注释：CInt 不同于 Fix 和 Int 函数删除数值的小数部分，而是采用四舍五入的方式。 当小数部分正好等于 0.5 时, CInt 总是将其四舍五入成最接近该数的偶数。例如， 0.5 四舍五入为 0, 以及 1.5 四舍五入为 2.

语法

```
CInt(expression)
```

参数	描述
expression	必需的。任何有效的表达式。

实例

例子 1

```
dim a
a=134.345
document.write(CInt(a))
```

输出：

```
134
```

例子 2

```
dim a
a=-30000.24
document.write(CInt(a))
```

输出：

```
-30000
```

VBScript 函数参考手册

VBScript CLng 函数

VBScript 函数参考手册

定义和用法

CLng 函数可把表达式转换为长整形（Long）类型。

注释：值必须是介于 -2147483648 与 2147483647 之间的数字。

注释：CLng 不同于 Fix 和 Int 函数删除小数部分， 而是采用四舍五入的方式。 当小数部分正好等于 0.5 时， CLng 函数总是将其四舍五入为最接近该数的偶数。如， 0.5 四舍五入为 0, 以及 1.5 四舍五入为 2 。

语法

```
CInt(expression)
```

参数	描述
expression	必需的。任何有效的表达式。

实例

例子 1

```
dim a,b
a=23524.45
b=23525.55
document.write(CLng(a))
document.write(CLng(b))
```

输出分别是：

```
23524
23526
```

VBScript 函数参考手册

VBScript CSng 函数

VBScript 函数参考手册

定义和用法

CSng 函数可把表达式转换为单精度（Single）类型。

注释：如果表达式在 Single 子类型允许的范围之外，则发生错误。

语法

```
CSng(expression)
```

参数	描述
expression	必需的。任何有效的表达式。

实例

例子 1

```
dim a,b
a=23524.4522
b=23525.5533
document.write(CSng(a) & "<br />")
document.write(CSng(b))
```

输出分别是：

```
23524.45
23525.55
```

VBScript 函数参考手册

VBScript CStr 函数

VBScript 函数参考手册

定义和用法

CStr 函数可把表达式转换为字符串（String）类型。

注释：若表达式的类型不同，那么 CStr 输出的结果也会有所不同。

语法

```
CStr(expression)
```

参数	描述
expression	必需的。任何有效的表达式。

当表达式为不同的值时，CStr 返回的结果：

表达式可能的值	CStr 相应的返回结果
Boolean	字符串，包含 True 或 False。
Date	字符串，包含系统的短日期格式日期。
Null	会发生 run-time 错误。
Empty	零长度字符串 ("")。
Error	字符串，包含跟随有错误号码的单词 Error。
其他数值	字符串，包含此数字。

实例

例子 1

```
dim a
a=false
document.write(CStr(a))
```

输出分别是：

```
false
```

例子 1

```
dim a
a=#01/01/01#
document.write(CStr(a))
```

输出分别是：

```
2001-1-1
```

VBScript 函数参考手册

VBScript Hex 函数

VBScript 函数参考手册

定义和用法

Hex 函数可返回一个表示指定数字的十六进制值的字符串。

注释：如果 number 参数不是整数，则在进行运算前将其四舍五入为最接近的整数。

语法

```
Hex(number)
```

参数	描述
expression	必需的。任何有效的表达式。如果 number 是： `Null` - 那么 Hex 函数会返回 Null。 `Empty` - 那么 Hex 函数会返回零 (0)。 其他数 - 那么 Hex 函数会返回最多 8 个十六进制字符。

实例

例子 1

```
document.write(Hex(3))
document.write(Hex(5))
document.write(Hex(9))
document.write(Hex(10))
document.write(Hex(11))
document.write(Hex(12))
document.write(Hex(400))
document.write(Hex(459))
document.write(Hex(460))
```

分别输出：

```
3
5
9
A
B
C
190
1CB
1CC
```

VBScript 函数参考手册

VBScript Oct 函数

VBScript 函数参考手册

定义和用法

Oct 函数可返回表示指定数字八进制值的字符串。

注释：如果 *number* 参数不是整数，则在进行运算前将其四舍五入为最接近的整数。

语法

```
Oct(_number_)
```

参数	描述
number	必需的。任何有效的表达式。如果 *number* 是： `Null` - 那么 Oct 函数会返回 Null。 `Empty` - 那么 Oct 函数会返回零 (0)。 其他数 - 那么 Oct 函数会返回最多 11 个十六进制字符。

实例

```
document.write(Oct(3))
document.write(Oct(5))
document.write(Oct(9))
document.write(Oct(10))
document.write(Oct(11))
document.write(Oct(12))
document.write(Oct(400))
document.write(Oct(459))
document.write(Oct(460))
```

分别输出：

```
3
5
11
12
13
14
620
713
714
```

亲自试一试

VBScript 函数参考手册

VBScript Format 函数

函数	描述
FormatCurrency	返回作为货币值进行格式化的表达式。
FormatDateTime	返回作为日期或时间进行格式化的表达式。
FormatNumber	返回作为数字进行格式化的表达式。
FormatPercent	返回作为百分数进行格式化的表达式。

VBScript FormatCurrency 函数

VBScript 函数参考手册

定义和用法

FormatCurrency 函数可返回作为货币值被格式化的表达式，使用系统控制面板中定义的货币符号。

语法

```
FormatCurrency(Expression[,NumDigAfterDec[,
IncLeadingDig[,UseParForNegNum[,GroupDig]]]])
```

参数	描述
expression	必需的。需被格式化的表达式。
NumDigAfterDec	指示小数点右侧显示位数的数值。默认值为 -1（使用的是计算机的区域设置）。
IncLeadingDig	可选的。指示是否显示小数值的前导零（leading zero）： -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False
UseParForNegNum	可选的。指示是否将负值置于括号中。 -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False
GroupDig	可选的。指示是否使用计算机区域设置中指定的数字分组符号将数字分组。 -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False

实例

例子 1

```
document.write(FormatCurrency(20000))
```

输出：

```
￥20,000.00
```

例子 2

```
document.write(FormatCurrency(20000.578,2))
```

输出：

```
￥20,000.58
```

例子 3

```
document.write(FormatCurrency(20000.578,2,,,0))
```

输出：

```
￥20000.58
```

VBScript 函数参考手册

VBScript FormatDateTime 函数

VBScript 函数参考手册

定义和用法

FormatDateTime 函数可格式化并返回一个合法的日期或时间表达式。

语法

```
FormatDateTime(date,format)
```

参数	描述
date	必需的。任何合法的日期表达式。(比如 Date() 或 Now())
format	可选的。规定所使用的日期/时间格式的格式值。

format 参数：

常数	值	描述
vbGeneralDate	0	显示日期和/或时间。如果有日期部分，则将该部分显示为短日期格式。如果有时间部分，则将该部分显示为长时间格式。如果都存在，则显示所有部分。
vbLongDate	1	使用计算机区域设置中指定的长日期格式显示日期。
vbShortDate	2	使用计算机区域设置中指定的短日期格式显示日期。
vbLongTime	3	使用此格式显示时间：hh:mm:ss PM/AM
vbShortTime	4	使用 24 小时格式 (hh:mm) 显示时间。

实例

例子 1

```
D = #2001/2/22#
document.write(FormatDateTime(D))
```

输出：

```
2001-2-22
```

例子 2

```
D = #2001/2/22#
document.write(FormatDateTime(D,1))
```

输出：

```
2001年2月22日
```

例子 3

```
D = #2001/2/22#
document.write(FormatDateTime(D,2))
```

输出：

```
2001-2-22
```

例子 4

```
D = #2001/2/22#
document.write(FormatDateTime(D,3))
```

输出：

```
2001-2-22
```

VBScript 函数参考手册

VBScript FormatNumber 函数

VBScript 函数参考手册

定义和用法

FormatNumber 函数可返回作为数字被格式化的表达式。

语法

```
FormatNumber(Expression[,NumDigAfterDec[,
IncLeadingDig[,UseParForNegNum[,GroupDig]]]])
```

参数	描述
expression	必需的。需被格式化的表达式。
NumDigAfterDec	指示小数点右侧显示位数的数值。默认值为 -1（使用的是计算机的区域设置）。
IncLeadingDig	可选的。指示是否显示小数值的前导零（leading zero）： -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False
UseParForNegNum	可选的。指示是否将负值置于括号中。 -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False
GroupDig	可选的。指示是否使用计算机区域设置中指定的数字分组符号将数字分组。 -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False

实例

例子 1

```
document.write(FormatNumber(20000))
```

输出：

```
20,000.00
```

例子 2

```
document.write(FormatNumber(20000.578,2))
```

输出：

```
20,000.58
```

例子 3

```
document.write(FormatNumber(20000.578,2,,,0))
```

输出：

```
20000.58
```

VBScript 函数参考手册

VBScript FormatPercent 函数

VBScript 函数参考手册

定义和用法

FormatPercent 函数可返回作为百分数被格式化的表达式（尾随有 % 符号的百分比（乘以 100））。

语法

```
FormatPercent(Expression[,NumDigAfterDec[,
IncLeadingDig[,UseParForNegNum[,GroupDig]]]])
```

参数	描述
expression	必需的。需被格式化的表达式。
NumDigAfterDec	指示小数点右侧显示位数的数值。默认值为 -1（使用的是计算机的区域设置）。
IncLeadingDig	可选的。指示是否显示小数值的前导零（leading zero）： -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False
UseParForNegNum	可选的。指示是否将负值置于括号中。 -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False
GroupDig	可选的。指示是否使用计算机区域设置中指定的数字分组符号将数字分组。 -2 = `TristateUseDefault` - 使用计算机区域设置中的设置。 -1 = `TristateTrue` - True 0 = `TristateFalse` - False

实例

例子 1

```
' 6 是 345 的百分之几?
document.write(FormatPercent(6/345))
```

输出：

```
1.74%
```

例子 2

```
' 6 是 345 的百分之几?
document.write(FormatPercent(6/345,1))
```

输出：

```
1.7%
```

VBScript 函数参考手册

VBScript Math 函数

函数	描述
Abs	返回指定数字的绝对值。
Atn	返回指定数字的反正切。
Cos	返回指定数字（角度）的余弦。
Exp	返回 e（自然对数的底）的幂次方。
Hex	返回指定数字的十六进制值。
Int	返回指定数字的整数部分。
Fix	返回指定数字的整数部分。
Log	返回指定数字的自然对数。
Oct	返回指定数字的余弦值。
Rnd	返回小于1但大于或等于0的一个随机数。
Sgn	返回可指示指定的数字的符号的一个整数。
Sin	返回指定数字（角度）的正弦。
Sqr	返回指定数字的平方根。
Tan	返回指定数字（角度）的正切。

VBScript Abs 函数

VBScript 函数参考手册

定义和用法

Abs 函数可返回指定的数字的绝对值。

注释：如果 number 参数包含Null，则返回 Null 。

注释：如果 number 参数是一个未初始化的变量，则返回 0 。

语法

```
Abs(number)
```

参数	描述
number	必需的。一个数值表达式。

实例

例子 1

```
document.write(Abs(1))
document.write(Abs(-1))
```

输出：

```
1
1
```

例子 2

```
document.write(Abs(48.4))
document.write(Abs(-48.4))
```

输出：

```
48.4
48.4
```

VBScript 函数参考手册

VBScript Atn 函数

VBScript 函数参考手册

定义和用法

Atn 函数可返回指定数字的正切。

语法

```
Atn(number)
```

参数	描述
number	必需的。一个数值表达式。

实例

例子 1

```
document.write(Atn(89))
```

输出：

```
1.55956084453693
```

例子 2

```
document.write(Atn(8.9))
```

输出：

```
1.45890606062322
```

例子 3

```
' 计算 pi 的值：
dim pi
pi=4*Atn(1)
document.write(pi)
```

输出：

```
3.14159265358979
```

VBScript 函数参考手册

VBScript Exp 函数

VBScript 函数参考手册

定义和用法

Exp 函数可e（自然对数的底）的幂次方。

注释：值不能超过 709.782712893 。常数 e 的值约为 2.718282。

提示：请参阅 Log 函数。Exp 函数完成 Log 函数的反运算，并且有时引用为反对数形式。

语法

```
Cos(number)
```

参数	描述
number	必需的。有效的数值表达式。

实例

例子 1

```
document.write(Exp(6.7))
```

输出：

```
812.405825167543
```

例子 2

```
document.write(Exp(-6.7))
```

输出：

```
1.23091190267348E-03
```

VBScript 函数参考手册

VBScript Int 函数

VBScript 函数参考手册

定义和用法

Int 函数可返回指定数字的整数部分。

注释：若 number 参数包含 Null，则返回 Null。

提示：请参阅 Fix 函数。Int 和 Fix 函数都删除 number 参数的小数部分并返回以整数表示的结果。

Int 和 Fix 函数的区别在于如果 number 参数为负数时，Int 函数返回小于或等于 number 的第一个负整数，而 Fix 函数返回大于或等于 number 参数的第一个负整数。例如，Int 将 -8.4 转换为 -9，而 Fix 函数将 -8.4 转换为 -8。

语法

```
Int(number)
```

参数	描述
number	必需的。有效的数值表达式。

实例

例子 1

```
document.write(Int(6.83227))
```

输出：

```
6
```

例子 2

```
document.write(Int(6.23443))
```

输出：

```
6
```

例子 3

```
document.write(Int(-6.13443))
```

输出：

```
-7
```

例子 4

```
document.write(Int(-6.93443))
```

输出：

```
-7
```

VBScript 函数参考手册

VBScript Fix 函数

VBScript 函数参考手册

定义和用法

Fix 函数可返回指定数字的整数部分。

注释：若 number 参数包含 Null，则返回 Null 。

提示：请参阅 Int 函数。Int 和 Fix 函数都删除 number 参数的小数部分并返回以整数表示的结果。

Int 和 Fix 函数的区别在于如果 number 参数为负数时，Int 函数返回小于或等于 number 的第一个负整数，而 Fix 函数返回大于或等于 number 参数的第一个负整数。例如，Int 将 -8.4 转换为 -9，而 Fix 函数将 -8.4 转换为 -8。

语法

```
Int(number)
```

参数	描述
number	必需的。有效的数值表达式。

实例

例子 1

```
document.write(Fix(6.83227))
```

输出：

```
6
```

例子 2

```
document.write(Fix(6.23443))
```

输出：

```
6
```

例子 3

```
document.write(Fix(-6.13443))
```

输出：

```
-6
```

例子 4

```
document.write(Fix(-6.93443))
```

输出：

```
-6
```

VBScript 函数参考手册

VBScript Log 函数

VBScript 函数参考手册

定义和用法

Log 函数可返回指定数据的自然对数。自然对数是以 e 为底的对数。

注释：不允许使用负值。

提示：请参阅 Exp 函数。

语法

```
Log(number)
```

参数	描述
number	必需的。大于 0 合法的数值表达式。

实例

例子 1

```
document.write(Log(38.256783227))
```

输出：

```
3.64432088381777
```

VBScript 函数参考手册

VBScript Rnd 函数

VBScript 函数参考手册

定义和用法

Rnd 函数可返回一个随机数。数字总是小于 **1** 但大于或等于 **0** 。

因每一次连续调用 Rnd 函数时都用序列中的前一个数作为下一个数的种子，所以对于任何最初给定的种子都会生成相同的数列。

在调用 Rnd 之前，先使用无参数的 Randomize 语句初始化随机数生成器，该生成器具有基于系统计时器的种子。

要产生指定范围的随机整数，请使用以下公式：

```
Int((upperbound - lowerbound + 1) * Rnd + lowerbound)
```

这里， upperbound 是此范围的上界，而 lowerbound 是此范围内的下界。

注释：要重复随机数的序列，请在使用数值参数调用 Randomize 之前，立即用负值参数调用 Rnd。使用同样 number 值的 Randomize 不能重复先前的随机数序列。

语法

```
Rnd[(number)]
```

参数	描述
number	可选的。合法的数值表达式。如果数字是： `<0` - Rnd 会每次都返回相同的值。 `>0` - Rnd 会返回序列中的下一个随机数。 `=0` - Rnd 返回最近生成的数。 省略 - Rnd 会返回序列中的下一个随机数。

实例

例子 1

```
document.write(Rnd)
```

输出：

```
0.7055475
```

例子 2

如果您使用例子 1 中的代码，相同的随机数会重复出现。

可以使用 Randomize 语句在页面每次重新载入的时候生成一个新的随机数：

```
Randomize
document.write(Rnd)
```

输出：

```
0.4758112
```

例子 3

```
dim max,min
max=100
min=1
document.write(Int((max-min+1)*Rnd+min))
```

输出：

```
71
```

VBScript 函数参考手册

VBScript Sgn 函数

VBScript 函数参考手册

定义和用法

Sgn 函数可返回指示指定数字的符号的整数。

语法

```
Sgn(number)
```

参数	描述
number	必需的。合法的数值表达式。如果数字是： `>0` - Sgn 会返回 1。 `<0` - Sgn 会返回 -1。 `=0` - Sgn 会返回 0。

实例

例子 1

```
document.write(Sgn(15))
```

输出：

```
1
```

例子 2

```
document.write(Sgn(-5.67))
```

输出：

```
-1
```

例子 3

```
document.write(Sgn(0))
```

输出：

```
0
```

VBScript 函数参考手册

VBScript Sin 函数

VBScript 函数参考手册

定义和用法

Sin 函数可返回指定数字（角度）的正弦。

Sin 函数取某个角并返回直角三角形两边的比值。此比值是直角三角形中该角的对边长度与斜边长度之比。结果的范围在 -1 到 1 之间。

注释：将角度乘以 pi/180 即可转换为弧度，将弧度乘以 180/pi 即可转换为角度。

语法

```
Sgn(number)
```

参数	描述
number	必需的。将某个角表示为弧度的有效表达式。

实例

例子 1

```
document.write(Sin(47))
```

输出：

```
0.123573122745224
```

例子 2

```
document.write(Sin(-47))
```

输出：

```
-0.123573122745224
```

VBScript 函数参考手册

VBScript Sqr 函数

VBScript 函数参考手册

定义和用法

Sqr 函数可返回一个数的平方根。

注释：number 参数不能是负值。

语法

```
Sqr(number)
```

参数	描述
number	必需的。大于等于 0 的有效数值表达式。

实例

例子 1

```
document.write(Sqr(9))
```

输出：

```
3
```

例子 2

```
document.write(Sqr(0))
```

输出：

```
0
```

例子 3

```
document.write(Sqr(47))
```

输出：

```
6.85565460040104
```

VBScript 函数参考手册

VBScript Tan 函数

VBScript 函数参考手册

定义和用法

Tan 函数可返回指定数字（角度）的正切。

Tan 取某个角并返回直角三角形两个直角边的比值。此比值是直角三角形中该角的对边长度与邻边长度之比。

注释：将角度乘以 pi /180 即可转换为弧度，将弧度乘以 180/pi 即可转换为角度。

语法

```
Sqr(number)
```

参数	描述
number	必需的。将某个角表示为弧度的有效表达式。

实例

例子 1

```
document.write(Tan(40))
```

输出：

```
-1.1172149309239
```

VBScript 函数参考手册

VBScript Array 函数

函数	描述
Array	返回一个包含数组的变量
Filter	返回下标从零开始的数组，其中包含基于特定过滤条件的字符串数组的子集。
IsArray	返回一个布尔值，可指示指定的变量是否是数组。
Join	返回一个由数组中若干子字符串组成的字符串。
LBound	返回指定数组维数的最小下标。
Split	返回下标从0开始的一维数组，包含指定数目的子字符串。
UBound	返回指定数组维数的最大下标。

VBScript Array 函数

VBScript 函数参考手册

定义和用法

Array 可返回一个包含数组的变量。

注释：数组中的第一个元素是零。

语法

```
Array(arglist)
```

参数	描述
arglist	必需的。数组中元素值的列表（由逗号分割）。

实例

例子 1

```
dim a
a=Array(5,10,15,20)
document.write(a(3))
```

输出：

```
20
```

例子 2

```
dim a
a=Array(5,10,15,20)
document.write(a(0))
```

输出：

```
5
```

VBScript 函数参考手册

VBScript Filter 函数

VBScript 函数参考手册

定义和用法

Filter 函数可返回一个基于 0 的数组，此数组包含以特定过滤条件为基础的字符串数组的子集。

注释：如果找不到与 value 参数相匹配的值，Filter 函数会返回一个空数组。

注释：若参数 inputstrings 为 Null 或者不是一维数组，则会发生错误。

语法

```
Filter(inputstrings,value[,include[,compare]])
```

参数	描述
inputstrings	必需的。需检索的一维字符串数组。
value	必需的。要搜索的字符串。
include	可选项。Boolean 值，指定返回的子字符串是否包含 Value。如果 Include 为 True，Filter 将返回包含子字符串 Value 的数组子集。如果 Include 为 False，Filter 将返回不包含子字符串 Value 的数组子集。
compare	可选的。规定所使用的字符串比较类型。

参数 compare 的值：

常数	值	描述
vbBinaryCompare	0	执行二进制比较。
vbTextCompare	1	执行文本比较。

实例

例子 1

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n")
document.write(b(0))
document.write(b(1))
document.write(b(2))
```

输出：

```
Sunday
Monday
Wednesday
```

例子 2

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n",false)
document.write(b(0))
document.write(b(1))
document.write(b(2))
```

输出：

```
Saturday
Tuesday
```

VBScript 函数参考手册

VBScript IsArray 函数

VBScript 函数参考手册

定义和用法

IsArray 函数可返回一个指示指定的变量是否为数组的布尔值。如果变量为数组，则返回 True，否则返回 False。

语法

```
IsArray(variable)
```

参数	描述
variable	必需的。任何变量。

实例

例子 1

```
dim a(5)
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
document.write(IsArray(a))
```

输出：

```
True
```

例子 2

```
dim a
a="Saturday"
document.write(IsArray(a))
```

输出：

```
False
```

VBScript 函数参考手册

VBScript Join 函数

VBScript 函数参考手册

定义和用法

Join 函数可返回一个由某个数组中一系列子字符串组成的字符串。

语法

```
Join(list[,delimiter])
```

参数	描述
list	必需的。一维数组，其中包含需被连接的子字符串。
delimiter	可选的。用于在返回的字符串中分割子字符串的字符。默认是空格字符。

实例

例子 1

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n")
document.write(join(b))
```

输出：

```
Sunday Monday Wednesday
```

例子 2

```
dim a(5),b
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
b=Filter(a,"n")
document.write(join(b,", "))
```

输出：

```
Sunday, Monday, Wednesday
```

VBScript 函数参考手册

VBScript LBound 函数

VBScript 函数参考手册

定义和用法

LBound 函数可返回指示数组维数的最小下标。

注释：任何维数的 LBound 总是 0。

提示：LBound 函数与 UBound 函数共同使用，可以确定数组的大小。使用 UBound 函数可以找到数组某一维的上界。

语法

```
LBound(arrayname[,dimension])
```

参数	描述
arrayname	必需的。数组变量的名称。
dimension	可选的。要返回哪一维的下界。 1 = 第一维， 2 = 第二维，以此类推。默认是 1 。

实例

例子 1

```
dim a(10)
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
a(5)="Thursday"
document.write(UBound(a))
document.write("<br />")
document.write(LBound(a))
```

输出：

```
10
0
```

VBScript 函数参考手册

VBScript Split 函数

VBScript 函数参考手册

定义和用法

Split 函数可返回基于 0 的一维数组，此数组包含指定数目的子字符串。

语法

```
Split(expression[,delimiter[,count[,compare]]])
```

参数	描述
expression	必需的。包含子字符串和分隔符的字符串表达式。
delimiter	可选的。用于识别子字符串界限的字符。默认是空格字符。
count	可选的。需被返回的子字符串的数目。-1 指示返回所有的子字符串。
compare	可选的。规定要使用的字符串比较类型。可采用下列的值： 0 = `vbBinaryCompare` - 执行二进制比较。 1 = `vbTextCompare` - 执行文本比较。

实例

例子 1

```
dim txt,a
txt="Hello World!"
a=Split(txt)
document.write(a(0) & "<br />")
document.write(a(1))
```

输出：

```
Hello
World!
```

VBScript 函数参考手册

VBScript UBound 函数

VBScript 函数参考手册

定义和用法

UBound 函数可返回指示数组维数的最大下标。

提示：LBound 函数与 UBound 函数共同使用，可以确定数组的大小。使用 UBound 函数可以找到数组某一维的上界。

语法

```
UBound(arrayname[,dimension])
```

参数	描述
arrayname	必需的。数组变量的名称。
dimension	可选的。要返回哪一维的上界。 1 = 第一维， 2 = 第二维，以此类推。默认是 1 。

实例

例子 1

```
dim a(10)
a(0)="Saturday"
a(1)="Sunday"
a(2)="Monday"
a(3)="Tuesday"
a(4)="Wednesday"
a(5)="Thursday"
document.write(UBound(a))
document.write("<br />")
document.write(LBound(a))
```

输出：

```
10
0
```

VBScript 函数参考手册

VBScript String 函数

函数	描述
InStr	返回字符串在另一字符串中首次出现的位置。检索从字符串的第一个字符开始。
InStrRev	返回字符串在另一字符串中首次出现的位置。检索从字符串的最末字符开始。
LCase	把指定字符串转换为小写。
Left	从字符串的左侧返回指定数目的字符。
Len	返回字符串中的字符数目。
LTrim	删除字符串左侧的空格。
RTrim	删除字符串右侧的空格。
Trim	删除字符串左侧和右侧的空格。
Mid	从字符串返回指定数目的字符。
Replace	使用另外一个字符串替换字符串的指定部分指定的次数。
Right	返回从字符串右侧开始指定数目的字符。
Space	返回由指定数目的空格组成的字符串。
StrComp	比较两个字符串，返回代表比较结果的一个值。
String	返回包含指定长度的重复字符的字符串。
StrReverse	反转字符串。
UCase	把指定的字符串转换为大写。

VBScript InStr 函数

VBScript 函数参考手册

定义和用法

InStr 函数可返回一个字符串在另一个字符串中首次出现的位置。

InStr 函数可返回下面的值：

- 如果 string1 为 ""（零长度） - InStr 返回 0
- 如果 string1 为 Null - InStr 返回 Null
- 如果 string2 为 "" - InStr 返回 start
- 如果 string2 为 Null - InStr 返回 Null
- 如果 string2 没有找到 - InStr 返回 0
- 如果在 string1 中找到 string2，InStr 返回找到匹配字符串的位置。
- 如果 start > Len(string1) - InStr 返回 0

提示：请参阅 InStrRev 函数。

语法

```
InStr([start,]string1,string2[,compare])
```

参数	描述
start	可选的。规定每次搜索的起始位置。默认是搜索起始位置是第一个字符。如果已规定 compare 参数，则必须有此参数。
string1	必需的。需要被搜索的字符串。
string2	必需的。需搜索的字符串。
compare	必需的。规定要使用的字符串比较类型。默认是 0 。可采用下列值： 0 = `vbBinaryCompare` - 执行二进制比较。 1 = `vbTextCompare` - 执行文本比较。

实例

例子 1

```
dim txt,pos
txt="This is a beautiful day!"
pos=InStr(txt,"his")
document.write(pos)
```

输出：

```
2
```

例子 2

```
dim txt,pos
txt="This is a beautiful day!"
'A textual comparison starting at position 4
pos=InStr(4,txt,"is",1)
document.write(pos)
```

输出：

```
6
```

例子 3

```
dim txt,pos
txt="This is a beautiful day!"
'A binary comparison starting at position 1
pos=InStr(1,txt,"B",0)
document.write(pos)
```

输出：

```
0
```

VBScript 函数参考手册

VBScript InStrRev 函数

VBScript 函数参考手册

定义和用法

InStrRev 函数可返回一个字符串在另一个字符串中首次出现的位置。搜索从字符串的末端开始，但是返回的位置是从字符串的起点开始计数的。

InStrRev 函数可返回下面的值：

- 如果 string1 为 ""（零长度） - InStr 返回 0
- 如果 string1 为 Null - InStr 返回 Null
- 如果 string2 为 "" - InStr 返回 start
- 如果 string2 为 Null - InStr 返回 Null
- 如果 string2 没有找到 - InStr 返回 0
- 如果在 string1 中找到 string2，InStr 返回找到匹配字符串的位置。
- 如果 start > Len(string1) - InStr 返回 0

提示：请参阅 InStr 函数。

语法

```
InStrRev(string1,string2[,start[,compare]])
```

参数	描述
start	可选的。规定每次搜索的起始位置。默认是搜索起始位置是第一个字符。如果已规定 compare 参数，则必须有此参数。
string1	必需的。需要被搜索的字符串。
string2	必需的。需搜索的字符串。
compare	必需的。规定要使用的字符串比较类型。默认是 0 。可采用下列值： 0 = `vbBinaryCompare` - 执行二进制比较。 1 = `vbTextCompare` - 执行文本比较。

实例

例子 1

```
dim txt,pos
txt="This is a beautiful day!"
pos=InStrRev(txt,"his")
document.write(pos)
```

输出：

```
2
```

例子 2

```
dim txt,pos
txt="This is a beautiful day!"
'textual comparison
pos=InStrRev(txt,"B",-1,1)
document.write(pos)
```

输出：

```
11
```

例子 3

```
dim txt,pos
txt="This is a beautiful day!"
'binary comparison
pos=InStrRev(txt,"T")
document.write(pos)
```

输出：

```
1
```

例子 4

```
dim txt,pos
txt="This is a beautiful day!"
'binary comparison
pos=InStrRev(txt,"t")
document.write(pos)
```

输出：

```
15
```

VBScript 函数参考手册

VBScript LCase 函数

VBScript 函数参考手册

定义和用法

LCase 函数可把指定的字符串转换为小写。

提示：请参阅 UCase 函数。

语法

```
LCase(string)
```

参数	描述
string	必需的。需要被转换为小写的字符串。

实例

例子 1

```
dim txt
txt="THIS IS A BEAUTIFUL DAY!"
document.write(LCase(txt))
```

输出：

```
this is a beautiful day!
```

例子 2

```
dim txt
txt="This Is a Beautiful Day!"
document.write(LCase(txt))
```

输出：

```
this is a beautiful day!
```

VBScript 函数参考手册

VBScript Left 函数

VBScript 函数参考手册

定义和用法

Left 函数可从字符串的左侧返回指定数目的字符。

提示：请使用 Len 函数来确定字符串中的字符数目。

提示：请参阅 Right 函数。

语法

```
Left(string,length)
```

参数	描述
string	必需的。从其中返回字符的字符串。
length	必需的。规定需返回多少字符。如果设置为 0，则返回空字符串("")。如果设置为大于或等于字符串的长度，则返回整个字符串。

实例

例子 1

```
dim txt
txt="This is a beautiful day!"
document.write(Left(txt,11))
```

输出：

```
This is a b
```

例子 2

```
dim txt
txt="This is a beautiful day!"
document.write(Left(txt,100))
```

输出：

```
This is a beautiful day!
```

例子 3

```
dim txt,x
txt="This is a beautiful day!"
x=Len(txt)
document.write(Left(txt,x))
```

输出：

```
This is a beautiful day!
```

VBScript 函数参考手册

VBScript Len 函数

VBScript 函数参考手册

定义和用法

Len 函数可返回字符串中字符的数目。

语法

```
Len(string|varname)
```

参数	描述
string	字符串表达式。
varname	变量名称。

实例

例子 1

```
dim txt
txt="This is a beautiful day!"
document.write(Len(txt))
```

输出：

```
24
```

例子 2

```
document.write(Len("This is a beautiful day!"))
```

输出：

```
24
```

VBScript 函数参考手册

VBScript LTrim 函数

VBScript 函数参考手册

定义和用法

LTrim 函数可删除字符串左侧的空格。

语法

```
Len(string|varname)
```

参数	描述
string	字符串表达式。

实例

例子 1

```
dim txt
txt="  This is a beautiful day!  "
document.write(LTrim(txt))
```

输出：

```
"This is a beautiful day!  "
```

VBScript 函数参考手册

VBScript Trim 函数

VBScript 函数参考手册

定义和用法

Trim 函数可删除字符串两侧的空格。

语法

```
Trim(string)
```

参数	描述
string	字符串表达式。

实例

例子 1

```
dim txt
txt="  This is a beautiful day!  "
document.write(Trim(txt))
```

输出：

```
"This is a beautiful day!"
```

VBScript 函数参考手册

VBScript Mid 函数

VBScript 函数参考手册

定义和用法

Mid 函数可从字符串中返回指定数目的字符。

提示：请使用 Len 函数来确定字符串中的字符数目。

语法

```
Mid(string,start[,length])
```

参数	描述
string	必需的。从其中返回字符的字符串表达式。如果字符串包含 Null，则返回 Null。
start	必需的。规定起始位置。如果设置为大于字符串中的字符数目，则返回空字符串("")。
length	可选的。要返回的字符数目。如果省略或 length 超过文本的字符数（包括 start 处的字符），将返回字符串中从 start 到字符串结束的所有字符。

实例

例子 1

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,1,1))
```

输出：

```
T
```

例子 2

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,1,11))
```

输出：

```
This is a b
```

例子 3

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,1))
```

输出：

```
This is a beautiful day!
```

例子 4

```
dim txt
txt="This is a beautiful day!"
document.write(Mid(txt,10))
```

输出：

```
beautiful day!
```

VBScript 函数参考手册

VBScript Replace 函数

VBScript 函数参考手册

定义和用法

Replace 函数可使用一个字符串替换另一个字符串指定的次数。

语法

```
Replace(string,find,replacewith[,start[,count[,compare]]])
```

参数	描述
string	必需的。需要被搜索的字符串。
find	必需的。将被替换的字符串部分。
replacewith	必需的。用于替换的子字符串。
start	可选的。规定开始位置。默认是 1。
count	可选的。规定指定替换的次数。默认是 -1，表示进行所有可能的替换。
compare	可选的。规定所使用的字符串比较类型。默认是 0。

参数 compare 的值：

常数	值	描述
vbBinaryCompare	0	执行二进制比较。
vbTextCompare	1	执行文本比较。

Replace 可能返回的值：

参数可能的取值	**Replace** 返回的值
expression 为零长度	零长度字符串 ("").
expression 为 Null	错误。
find 参数为零长度	expression 的副本。
replacewith 参数为零长度	expression 的副本，其中删除了所有由 find 参数指定的内容。
start > Len(expression)	零长度字符串。
count 为 0	expression 的副本。

实例

例子 1

```
dim txt
txt="This is a beautiful day!"
document.write(Replace(txt,"beautiful","horrible"))
```

输出：

```
This is a horrible day!
```

VBScript 函数参考手册

VBScript Right 函数

VBScript 函数参考手册

定义和用法

Right 函数可从字符串右侧返回指定数目的字符。

提示：请使用Len 函数来确定字符串中的字符数目。

提示：请参阅 Left 函数。

语法

```
Right(string,length)
```

参数	描述
string	必需的。从其中返回字符的字符串。
length	必需的。规定返回多少字符。如果设置为 0，则返回空字符串 ("")。如果设置为大于或等于字符串的长度，则返回整个的字符串。

实例

例子 1

```
dim txt
txt="This is a beautiful day!"
document.write(Right(txt,11))
```

输出：

```
utiful day!
```

例子 2

```
dim txt
txt="This is a beautiful day!"
document.write(Right(txt,100))
```

输出：

```
This is a beautiful day!
```

例子 3

```
dim txt,x
txt="This is a beautiful day!"
x=Len(txt)
document.write(Right(txt,x))
```

输出：

```
This is a beautiful day!
```

VBScript 函数参考手册

VBScript Space 函数

VBScript 函数参考手册

定义和用法

Space 函数可返回一个由指定数目的空格组成的字符串。

语法

```
Space(number)
```

参数	描述
number	必需的。字符串中的空格数目。

实例

例子 1

```
dim txt
txt=Space(10)
document.write(txt)
```

输出：

```
"          "
```

VBScript 函数参考手册

VBScript StrComp 函数

VBScript 函数参考手册

定义和用法

StrComp 函数可比较两个字符串，并返回表示比较结果的一个值。

StrComp 函数可返回下面的值：

- -1 (如果 string1 < string2)
- 0 (如果 string1 = string2)
- 1 (如果 string1 > string2)
- Null (如果 string1 或 string2 为 Null)

语法

```
StrComp(string1,string2[,compare])
```

参数	描述
string1	必需的。字符串表达式。
string2	必需的。字符串表达式。
compare	可选的。规定要使用的字符串比较类型。默认是 0。Optional. Specifies the string comparison to use. Default is 0可采用的值： 0 = `vbBinaryCompare` - 执行二进制比较 1 = `vbTextCompare` - 执行文本比较

实例

例子 1

```
document.write(StrComp("VBScript","VBScript"))
```

输出：

```
0
```

例子 2

```
document.write(StrComp("VBScript","vbscript"))
```

输出：

```
-1
```

例子 3

```
document.write(StrComp("VBScript","vbscript",1))
```

输出：

```
0
```

VBScript 函数参考手册

VBScript String 函数

VBScript 函数参考手册

定义和用法

String 函数可返回包含指定长度的重复字符的一个字符串。

语法

```
String(number,character)
```

参数	描述
number	必需的。被返回字符串的长度。
character	必需的。被重复的字符。

实例

例子 1

```
document.write(String(10,"#"))
```

输出：

```
##########
```

例子 2

```
document.write(String(4,"*"))
```

输出：

```
****
```

例子 3

```
document.write(String(4,42))
```

输出：

```
****
```

例子 4

```
document.write(String(4,"XYZ"))
```

输出：

```
XXXX
```

VBScript 函数参考手册

VBScript StrReverse 函数

VBScript 函数参考手册

定义和用法

StrReverse 函数可反转一个字符串。

语法

```
StrReverse(string)
```

参数	描述
string	必需的。需被反转的字符串。

实例

例子 1

```
dim txt
txt="This is a beautiful day!"
document.write(StrReverse(txt))
```

输出：

```
!yad lufituaeb a si sihT
```

VBScript 函数参考手册

VBScript UCase 函数

VBScript 函数参考手册

定义和用法

UCase 函数可把指定的字符串转换为大写。

提示：请参阅 LCase 函数。

语法

```
UCase(string)
```

参数	描述
string	必需的。需被转换为大写的字符串。

实例

例子 1

```
dim txt
txt="This is a beautiful day!"
document.write(UCase(txt))
```

输出：

```
THIS IS A BEAUTIFUL DAY!
```

例子 2

```
dim txt
txt="This Is a Beautiful Day!"
document.write(UCase(txt))
```

输出：

```
THIS IS A BEAUTIFUL DAY!
```

VBScript 函数参考手册

VBScript 其他函数

函数	描述
CreateObject	创建指定类型对象。
Eval	计算表达式，并返回结果。
GetLocale	返回当前区域设置 ID 值。
GetObject	返回对文件中 automation 对象的引用。
GetRef	允许您把 VBScript 子程序连接到页面上的一个 DHTML 事件。
InputBox	可显示对话框，用户可在其中输入文本，并/或点击按钮，然后返回结果。
IsEmpty	返回一个布尔值，指示指定的变量是否已被初始化。
IsNull	返回一个布尔值，指示指定的变量是否包含无效数据 (Null)。
IsNumeric	返回一个布尔值，指示指定的表达式是否可作为数字来计算。
IsObject	返回一个布尔值，指示指定的表达式是否是一个 automation 对象。
LoadPicture	返回一个图片对象。仅用于32位平台。
MsgBox	显示消息框，等待用户点击按钮，并返回指示用户点击了哪个按钮的值。
RGB	返回一个表示 RGB 颜色值的数字。
Round	对数进行四舍五入。
ScriptEngine	返回使用中的脚本语言。
ScriptEngineBuildVersion	返回使用中的脚本引擎版本号。
ScriptEngineMajorVersion	返回使用中的脚本引擎的主版本号。
ScriptEngineMinorVersion	返回使用中的脚本引擎的次版本号。
SetLocale	设置地区 ID ，并返回之前的地区 ID。
TypeName	返回指定变量的子类型。
VarType	返回指示变量子类型的值。

VBScript UCase 函数

VBScript 函数参考手册

定义和用法

CreateObject 函数可创建一个指定类型的对象。

语法

```
CreateObject(servername.typename[,location])
```

参数	描述
servername	必需的。提供此对象的应用程序名称。
typename	必需的。对象的类型或类（type/class）。
location	可选的。在何处创建对象。

实例

例子 1

```
dim myexcel
Set myexcel=CreateObject("Excel.Sheet")

myexcel.Application.Visible=True

...code...

myexcel.Application.Quit

Set myexcel=Nothing
```

VBScript 函数参考手册

VBScript GetLocale 函数

VBScript 函数参考手册

定义和用法

GetLocale 函数可返回当前的 locale ID 。

locale 是用户参考信息集合，比如用户的语言、国家和文化传统。locale 决定键盘布局、字母排序顺序和日期、时间、数字与货币格式等等。

返回值可以是任意一个 32-位 的值，如 区域设置 ID 所示。

语法

```
GetLocale()
```

实例

例子 1

```
dim c
c=GetLocale
document.write(c)
```

输出：

```
1033
```

Locale ID 表

Locale 描述	简写	十六进制值	十进制值
Afrikaans	af	0x0436	1078
Albanian	sq	0x041C	1052
Arabic ?United Arab Emirates	ar-ae	0x3801	14337
Arabic - Bahrain	ar-bh	0x3C01	15361

Arabic - Algeria	ar-dz	0x1401	5121
Arabic - Egypt	ar-eg	0x0C01	3073
Arabic - Iraq	ar-iq	0x0801	2049
Arabic - Jordan	ar-jo	0x2C01	11265
Arabic - Kuwait	ar-kw	0x3401	13313
Arabic - Lebanon	ar-lb	0x3001	12289
Arabic - Libya	ar-ly	0x1001	4097
Arabic - Morocco	ar-ma	0x1801	6145
Arabic - Oman	ar-om	0x2001	8193
Arabic - Qatar	ar-qa	0x4001	16385
Arabic - Saudi Arabia	ar-sa	0x0401	1025
Arabic - Syria	ar-sy	0x2801	10241
Arabic - Tunisia	ar-tn	0x1C01	7169
Arabic - Yemen	ar-ye	0x2401	9217
Armenian	hy	0x042B	1067
Azeri ?Latin	az-az	0x042C	1068
Azeri ?Cyrillic	az-az	0x082C	2092
Basque	eu	0x042D	1069
Belarusian	be	0x0423	1059
Bulgarian	bg	0x0402	1026
Catalan	ca	0x0403	1027
Chinese - China	zh-cn	0x0804	2052
Chinese - Hong Kong S.A.R.	zh-hk	0x0C04	3076
Chinese ?Macau S.A.R	zh-mo	0x1404	5124
Chinese - Singapore	zh-sg	0x1004	4100
Chinese - Taiwan	zh-tw	0x0404	1028
Croatian	hr	0x041A	1050
Czech	cs	0x0405	1029
Danish	da	0x0406	1030
Dutch ?The Netherlands	nl-nl	0x0413	1043
Dutch - Belgium	nl-be	0x0813	2067
English - Australia	en-au	0x0C09	3081

English - Belize	en-bz	0x2809	10249
English - Canada	en-ca	0x1009	4105
English ?Carribbean	en-cb	0x2409	9225
English - Ireland	en-ie	0x1809	6153
English - Jamaica	en-jm	0x2009	8201
English - New Zealand	en-nz	0x1409	5129
English ?Phillippines	en-ph	0x3409	13321
English - South Africa	en-za	0x1C09	7177
English - Trinidad	en-tt	0x2C09	11273
English - United Kingdom	en-gb	0x0809	2057
English - United States	en-us	0x0409	1033
Estonian	et	0x0425	1061
Farsi	fa	0x0429	1065
Finnish	fi	0x040B	1035
Faroese	fo	0x0438	1080
French - France	fr-fr	0x040C	1036
French - Belgium	fr-be	0x080C	2060
French - Canada	fr-ca	0x0C0C	3084
French - Luxembourg	fr-lu	0x140C	5132
French - Switzerland	fr-ch	0x100C	4108
Gaelic ?Ireland	gd-ie	0x083C	2108
Gaelic - Scotland	gd	0x043C	1084
German - Germany	de-de	0x0407	1031
German - Austria	de-at	0x0C07	3079
German - Liechtenstein	de-li	0x1407	5127
German - Luxembourg	de-lu	0x1007	4103
German - Switzerland	de-ch	0x0807	2055
Greek	el	0x0408	1032
Hebrew	he	0x040D	1037
Hindi	hi	0x0439	1081
Hungarian	hu	0x040E	1038

Icelandic	is	0x040F	1039
Indonesian	id	0x0421	1057
Italian - Italy	it-it	0x0410	1040
Italian - Switzerland	it-ch	0x0810	2064
Japanese	ja	0x0411	1041
Korean	ko	0x0412	1042
Latvian	lv	0x0426	1062
Lithuanian	lt	0x0427	1063
FYRO Macedonian	mk	0x042F	1071
Malay - Malaysia	ms-my	0x043E	1086
Malay ?Brunei	ms-bn	0x083E	2110
Maltese	mt	0x043A	1082
Marathi	mr	0x044E	1102
Norwegian - Bokm錶	no-no	0x0414	1044
Norwegian ?Nynorsk	no-no	0x0814	2068
Polish	pl	0x0415	1045
Portuguese - Portugal	pt-pt	0x0816	2070
Portuguese - Brazil	pt-br	0x0416	1046
Raeto-Romance	rm	0x0417	1047
Romanian - Romania	ro	0x0418	1048
Romanian - Moldova	ro-mo	0x0818	2072
Russian	ru	0x0419	1049
Russian - Moldova	ru-mo	0x0819	2073
Sanskrit	sa	0x044F	1103
Serbian - Cyrillic	sr-sp	0x0C1A	3098
Serbian ?Latin	sr-sp	0x081A	2074
Setsuana	tn	0x0432	1074
Slovenian	sl	0x0424	1060
Slovak	sk	0x041B	1051
Sorbian	sb	0x042E	1070
Spanish - Spain	es-es	0x0C0A	1034
Spanish - Argentina	es-ar	0x2C0A	11274

Spanish - Bolivia	es-bo	0x400A	16394
Spanish - Chile	es-cl	0x340A	13322
Spanish - Colombia	es-co	0x240A	9226
Spanish - Costa Rica	es-cr	0x140A	5130
Spanish - Dominican Republic	es-do	0x1C0A	7178
Spanish - Ecuador	es-ec	0x300A	12298
Spanish - Guatemala	es-gt	0x100A	4106
Spanish - Honduras	es-hn	0x480A	18442
Spanish - Mexico	es-mx	0x080A	2058
Spanish - Nicaragua	es-ni	0x4C0A	19466
Spanish - Panama	es-pa	0x180A	6154
Spanish - Peru	es-pe	0x280A	10250
Spanish - Puerto Rico	es-pr	0x500A	20490
Spanish - Paraguay	es-py	0x3C0A	15370
Spanish - El Salvador	es-sv	0x440A	17418
Spanish - Uruguay	es-uy	0x380A	14346
Spanish - Venezuela	es-ve	0x200A	8202
Sutu	sx	0x0430	1072
Swahili	sw	0x0441	1089
Swedish - Sweden	sv-se	0x041D	1053
Swedish - Finland	sv-fi	0x081D	2077
Tamil	ta	0x0449	1097
Tatar	tt	0X0444	1092
Thai	th	0x041E	1054
Turkish	tr	0x041F	1055
Tsonga	ts	0x0431	1073
Ukrainian	uk	0x0422	1058
Urdu	ur	0x0420	1056
Uzbek ?Cyrillic	uz-uz	0x0843	2115
Uzbek ?Latin	uz-uz	0x0443	1091
Vietnamese	vi	0x042A	1066

Xhosa	xh	0x0434	1076
Yiddish	yi	0x043D	1085
Zulu	zu	0x0435	1077

VBScript 函数参考手册

VBScript GetObject 函数

VBScript 函数参考手册

定义和用法

GetObject 函数可返回对文件中 Automation 对象的引用。

语法

```
GetObject([pathname][,class])
```

参数	描述
pathname	可选的。包含 automation 对象的文件的完整路径和名称。如果此参数被忽略，就必须有 class 参数。
class	可选的。automation 对象的类。此参数使用此语法：appname.objectype。

VBScript 函数参考手册

VBScript GetRef 函数

VBScript 函数参考手册

定义和用法

GetRef 函数可把一段 VBScript 过程（子程序）连接到页面的一个 DHTML 事件上。

语法

```
Set object.event=GetRef(procname)
```

参数	描述
object	必需的。事件所关联的对象的名称。
event	Required. 要与函数绑定的事件的名称。
procname	Required. 与事件关联的 Sub 或 Function 过程的名称。

实例

```
Function test()
dim txt
txt="GetRef Test" & vbCrLf
txt=txt & "Hello World!"
MsgBox txt
End Function

Window.Onload=GetRef("test")
```

VBScript 函数参考手册

VBScript InputBox 函数

VBScript 函数参考手册

定义和用法

InputBox 函数可显示一个对话框，用户可在其中输入文本并/或点击一个按钮。如果用户点击点击确认按钮或按键盘上的回车键， 则 InputBox 函数返回文本框中的文本。如果用户点击取消按钮，函数返回一个空字符串("")。

注释：若同时规定helpfile 和 context 参数，则会向对话框添加一个帮助按钮。

提示：请参阅 MsgBox 函数。

语法

```
InputBox(prompt[,title][,default][,xpos][,ypos][,helpfile,context])
```

参数	描述
prompt	必需的。现实在对话框中的消息。prompt 的最大长度大约是 1024 个字符，这取决于所使用的字符的宽度。如果 prompt 中包含多个行，则可在各行之间用回车符 (Chr(13))、换行符 (Chr(10)) 或回车换行符的组合 (Chr(13) & Chr(10)) 以分隔各行。
title	可选的。显示在对话框标题栏中的字符串表达式。如果省略 title，则应用程序的名称将显示在标题栏中。
default	可选的。显示在文本框中的字符串表达式，在没有其它输入时作为默认的响应值。如果省略 default，则文本框为空。
xpos	可选的。数值表达式，用于指定对话框的左边缘与屏幕左边缘的水平距离（单位为缇）。如果省略 xpos，则对话框会在水平方向居中。
ypos	可选的。数值表达式，用于指定对话框的上边缘与屏幕上边缘的垂直距离（单位为缇）。如果省略 ypos，则对话框显示在屏幕垂直方向距下边缘大约三分之一处。
helpfile	可选的。字符串表达式，用于标识为对话框提供上下文相关帮助的帮助文件。如果已提供 helpfile，则必须提供 context。
context	可选的。数值表达式，用于标识由帮助文件的作者指定给某个帮助主题的上下文编号。如果已提供 context，则必须提供 helpfile。

实例

```
dim fname
fname=InputBox("Enter your name:")
MsgBox("Your name is " & fname)
```

VBScript 函数参考手册

VBScript IsEmpty 函数

VBScript 函数参考手册

定义和用法

IsEmpty 函数可返回指定的变量是否被初始化的布尔值。如果未被初始化，则返回 true ，否则返回 False 。

语法

```
IsEmpty(expression)
```

参数	描述
expression	必需的。表达式（通常是一个变量名）。

实例

```
dim x
document.write(IsEmpty(x))
x=10
document.write(IsEmpty(x))
x=Empty
document.write(IsEmpty(x))
x=Null
document.write(IsEmpty(x))
```

分别输出：

```
True
False
True
False
```

VBScript 函数参考手册

VBScript IsNull 函数

VBScript 函数参考手册

定义和用法

IsNull 函数可返回指示指定表达式是否无效数据的布尔值。如果表达式是 Null，则返回 True，否则返回 False。

语法

```
IsNull(expression)
```

参数	描述
expression	必需的。表达式。

实例

```
dim x
document.write(IsNull(x))
x=10
document.write(IsNull(x))
x=Empty
document.write(IsNull(x))
x=Null
document.write(IsNull(x))
```

分别输出：

```
False
False
False
True
```

VBScript 函数参考手册

VBScript IsNumeric 函数

VBScript 函数参考手册

定义和用法

IsNumeric 函数可返回指示指定的表达式是否可作为数字来计算的布尔值。如果作为数字来计算，则返回 True ，否则返回 False。

注释：如果表达式是日期表达式，则 IsNumeric 返回 False。

语法

```
IsNumeric(expression)
```

参数	描述
expression	必需的。表达式。

实例

```
dim x
x=10
document.write(IsNumeric(x))
x=Empty
document.write(IsNumeric(x))
x=Null
document.write(IsNumeric(x))
x="10"
document.write(IsNumeric(x))
x="911 Help"
document.write(IsNumeric(x))
```

分别输出：

```
True
True
False
True
False
```

VBScript 函数参考手册

VBScript IsObject 函数

VBScript 函数参考手册

定义和用法

IsObject 函数可返回指示是否指定的表达式是一个 automation 对象的布尔值。如果表达式是对象，则返回 True 。否则返回 False。

语法

```
IsObject(expression)
```

参数	描述
expression	必需的。表达式。

实例

例子 1

```
dim x
set x=me
document.write(IsObject(x))
```

输出：

```
True
```

例子 2

```
dim x
x="me"
document.write(IsObject(x))
```

输出：

```
False
```

VBScript 函数参考手册

VBScript LoadPicture 函数

VBScript 函数参考手册

定义和用法

LoadPicture 函数可返回一个图片对象。

可被 LoadPicture 函数识别的图形格式有：

- bitmap 文件 (.bmp)
- icon 文件 (.ico)
- run-length encoded 文件 (.rle)
- metafile 文件 (.wmf)
- enhanced meta文件 (.emf)
- GIF 文件 (.gif)
- JPEG 文件 (.jpg)

注释：此函数仅供 32-位 平台。

语法

```
LoadPicture(picturename)
```

参数	描述
picturename	必需的。需被载入的图片文件的文件名。

VBScript 函数参考手册

VBScript MsgBox 函数

VBScript 函数参考手册

定义和用法

MsgBox 函数可显示一个消息框，等待用户点击某个按钮，然后返回指示被点击按钮的值。

MsgBox 函数可返回下面的值：

- 1 = vbOK - 确定按钮被单击。
- 2 = vbCancel - 取消按钮被单击。
- 3 = vbAbort - 终止按钮被单击。
- 4 = vbRetry - 重试按钮被单击。
- 5 = vbIgnore - 忽略按钮被单击。
- 6 = vbYes - 是按钮被单击。
- 7 = vbNo - 否按钮被单击。

注释：当 helpfile 和 context 参数均被规定后，用户可按 F1 键来查看帮助。

提示：请参阅 InputBox 函数。

语法

```
MsgBox(prompt[,buttons][,title][,helpfile,context])
```

参数	描述
prompt	必需的。作为消息显示在对话框中的字符串表达式。prompt 的最大长度大约是 1024 个字符，这取决于所使用的字符的宽度。如果 prompt 中包含多个行，则可在各行之间用回车符 (Chr(13))、换行符 (Chr(10)) 或回车换行符的组合 (Chr(13) & Chr(10)) 分隔各行。
buttons	数值表达式，是表示指定显示按钮的数目和类型、使用的图标样式，默认按钮的标识以及消息框样式的数值的总和。如果省略，则 buttons 的默认值为 0。button 的取值： 0 = `vbOKOnly` - 只显示确定按钮。 1 = `vbOKCancel` - 显示确定和取消按钮。 2 = `vbAbortRetryIgnore` - 显示放弃、重试和忽略按钮。 3 = `vbYesNoCancel` - 显示是、否和取消按钮。 4 = `vbYesNo` - 显示是和否按钮。 5 = `vbRetryCancel` - 显示重试和取消按钮。 16 = `vbCritical` - 显示临界信息图标。 32 = `vbQuestion` - 显示警告查询图标。 48 = `vbExclamation` - 显示警告消息图标。 64 = `vbInformation` - 显示信息消息图标。 0 = `vbDefaultButton1` - 第一个按钮为默认按钮。 256 = `vbDefaultButton2` - 第二个按钮为默认按钮。 512 = `vbDefaultButton3` - 第三个按钮为默认按钮。 768 = `vbDefaultButton4` - 第四个按钮为默认按钮。 0 = `vbApplicationModal` - 应用程序模式：用户必须响应消息框才能继续在当前应用程序中工作。 4096 = `vbSystemModal` - 系统模式：在用户响应消息框前，所有应用程序都被挂起。第一组值 (0 - 5) 用于描述对话框中显示的按钮类型与数目；第二组值 (16, 32, 48, 64) 用于描述图标的样式；第三组值 (0, 256, 512) 用于确定默认按钮；而第四组值 (0, 4096) 则决定消息框的样式。在将这些数字相加以生成 buttons 参数值时，只能从每组值中取用一个数字。
title	显示在对话框标题栏中的字符串表达式。如果省略 title，则将应用程序的名称显示在标题栏中。
helpfile	字符串表达式，用于标识为对话框提供上下文相关帮助的帮助文件。如果已提供 helpfile，则必须提供 context。在 16 位系统平台上不可用。
context	数值表达式，用于标识由帮助文件的作者指定给某个帮助主题的上下文编号。如果已提供 context，则必须提供 helpfile。在 16 位系统平台上不可用。

实例

```
dim answer
answer=MsgBox("Hello everyone!",65,"Example")
document.write(answer)
```

VBScript 函数参考手册

VBScript RGB 函数

VBScript 函数参考手册

定义和用法

RGB 函数可返回表示 RGB 颜色值的数字。

语法

```
RGB(red,green,blue)
```

参数	描述
red	必需的。介于 0 - 255 之间（且包括）的数字，代表颜色红色部分。
green	必需的。介于 0 - 255 之间（且包括）的数字，代表颜色绿色部分。
blue	必需的。介于 0 - 255 之间（且包括）的数字，代表颜色蓝色部分。

实例

例子 1

```
document.write(rgb(255,0,0))
```

输出：

```
255
```

例子 2

```
document.write(rgb(255,30,30))
```

输出：

```
1974015
```

VBScript 函数参考手册

VBScript Round 函数

VBScript 函数参考手册

定义和用法

Round 函数可对数字进行四舍五入。

语法

```
Round(expression[,numdecimalplaces])
```

参数	描述
expression	必需的。需要被四舍五入的表达式。
numdecimalplaces	可选的。规定对小数点右边的多少位进行四舍五入。默认是 0。

实例

例子 1

```
dim x
x=24.13278
document.write(Round(x))
```

输出：

```
24
```

例子 2

```
dim x
x=24.13278
document.write(Round(x,2))
```

输出：

```
24.13
```

VBScript 函数参考手册

VBScript ScriptEngine, ScriptEngineBuildVersion, ScriptEngineMajorVersion, 以及 ScriptEngineMinorVersion 函数

VBScript 函数参考手册

ScriptEngine 函数

ScriptEngine 函数可返回当前使用的脚本语言。

此函数可返回下面的字符串：

- VBScript - 指示当前使用的脚本引擎是 Microsoft(R) Visual Basic(R) Scripting Edition
- JScript - 指示当前使用的编写脚本引擎是 Microsoft Visual Basic Scripting Edition(R)。
- VBA - 指示当前使用的脚本引擎是 Microsoft Visual Basic for Applications。

ScriptEngineBuildVersion 函数

ScriptEngineBuildVersion 函数可返回当前在用的脚本引擎的版本号。

ScriptEngineMajorVersion 函数

ScriptEngineMajorVersion 函数可返回当前在用的脚本引擎的主版本号。

ScriptEngineMinorVersion 函数

ScriptEngineMinorVersion 函数可返回当前在用的脚本引擎的副版本。

语法

```
ScriptEngine

ScriptEngineBuildVersion

ScriptEngineMajorVersion

ScriptEngineMinorVersion
```

实例

```
document.write(ScriptEngine)
document.write(ScriptEngineBuildVersion)
document.write(ScriptEngineMajorVersion)
document.write(ScriptEngineMinorVersion)
```

输出：

```
VBScript
8827
5
6
```

VBScript 函数参考手册

VBScript SetLocale 函数

VBScript 函数参考手册

定义和用法

SetLocale 函数可设置 locale ID，并返回之前的 locale ID。

locale 是用户参考信息集合，比如用户的语言、国家和文化传统。locale 可决定键盘布局、字母排序顺序和日期、时间、数字与货币格式等等。

语法

```
SetLocale(lcid)
```

参数	描述
lcid	必需的。任意一个在 Locale ID 表 中的短字符串、十六进制值、十进制值，该值必须唯一标识一个地理区域。如果 lcid 参数被设置为 0， 则 locale 将由系统设置。

实例

```
document.write(SetLocale(2057))
document.write(SetLocale(2058))
```

输出：

```
1033
2057
```

Locale ID 表

Locale 描述	简写	十六进制值	十进制值
Afrikaans	af	0x0436	1078
Albanian	sq	0x041C	1052
Arabic ?United Arab Emirates	ar-ae	0x3801	14337

Arabic - Bahrain	ar-bh	0x3C01	15361
Arabic - Algeria	ar-dz	0x1401	5121
Arabic - Egypt	ar-eg	0x0C01	3073
Arabic - Iraq	ar-iq	0x0801	2049
Arabic - Jordan	ar-jo	0x2C01	11265
Arabic - Kuwait	ar-kw	0x3401	13313
Arabic - Lebanon	ar-lb	0x3001	12289
Arabic - Libya	ar-ly	0x1001	4097
Arabic - Morocco	ar-ma	0x1801	6145
Arabic - Oman	ar-om	0x2001	8193
Arabic - Qatar	ar-qa	0x4001	16385
Arabic - Saudi Arabia	ar-sa	0x0401	1025
Arabic - Syria	ar-sy	0x2801	10241
Arabic - Tunisia	ar-tn	0x1C01	7169
Arabic - Yemen	ar-ye	0x2401	9217
Armenian	hy	0x042B	1067
Azeri ?Latin	az-az	0x042C	1068
Azeri ?Cyrillic	az-az	0x082C	2092
Basque	eu	0x042D	1069
Belarusian	be	0x0423	1059
Bulgarian	bg	0x0402	1026
Catalan	ca	0x0403	1027
Chinese - China	zh-cn	0x0804	2052
Chinese - Hong Kong S.A.R.	zh-hk	0x0C04	3076
Chinese ?Macau S.A.R	zh-mo	0x1404	5124
Chinese - Singapore	zh-sg	0x1004	4100
Chinese - Taiwan	zh-tw	0x0404	1028
Croatian	hr	0x041A	1050
Czech	cs	0x0405	1029
Danish	da	0x0406	1030
Dutch ?The Netherlands	nl-nl	0x0413	1043

Dutch - Belgium	nl-be	0x0813	2067
English - Australia	en-au	0x0C09	3081
English - Belize	en-bz	0x2809	10249
English - Canada	en-ca	0x1009	4105
English ?Carribbean	en-cb	0x2409	9225
English - Ireland	en-ie	0x1809	6153
English - Jamaica	en-jm	0x2009	8201
English - New Zealand	en-nz	0x1409	5129
English ?Phillippines	en-ph	0x3409	13321
English - South Africa	en-za	0x1C09	7177
English - Trinidad	en-tt	0x2C09	11273
English - United Kingdom	en-gb	0x0809	2057
English - United States	en-us	0x0409	1033
Estonian	et	0x0425	1061
Farsi	fa	0x0429	1065
Finnish	fi	0x040B	1035
Faroese	fo	0x0438	1080
French - France	fr-fr	0x040C	1036
French - Belgium	fr-be	0x080C	2060
French - Canada	fr-ca	0x0C0C	3084
French - Luxembourg	fr-lu	0x140C	5132
French - Switzerland	fr-ch	0x100C	4108
Gaelic ?Ireland	gd-ie	0x083C	2108
Gaelic - Scotland	gd	0x043C	1084
German - Germany	de-de	0x0407	1031
German - Austria	de-at	0x0C07	3079
German - Liechtenstein	de-li	0x1407	5127
German - Luxembourg	de-lu	0x1007	4103
German - Switzerland	de-ch	0x0807	2055
Greek	el	0x0408	1032
Hebrew	he	0x040D	1037
Hindi	hi	0x0439	1081

Hungarian	hu	0x040E	1038
Icelandic	is	0x040F	1039
Indonesian	id	0x0421	1057
Italian - Italy	it-it	0x0410	1040
Italian - Switzerland	it-ch	0x0810	2064
Japanese	ja	0x0411	1041
Korean	ko	0x0412	1042
Latvian	lv	0x0426	1062
Lithuanian	lt	0x0427	1063
FYRO Macedonian	mk	0x042F	1071
Malay - Malaysia	ms-my	0x043E	1086
Malay ?Brunei	ms-bn	0x083E	2110
Maltese	mt	0x043A	1082
Marathi	mr	0x044E	1102
Norwegian - Bokm錶	no-no	0x0414	1044
Norwegian ?Nynorsk	no-no	0x0814	2068
Polish	pl	0x0415	1045
Portuguese - Portugal	pt-pt	0x0816	2070
Portuguese - Brazil	pt-br	0x0416	1046
Raeto-Romance	rm	0x0417	1047
Romanian - Romania	ro	0x0418	1048
Romanian - Moldova	ro-mo	0x0818	2072
Russian	ru	0x0419	1049
Russian - Moldova	ru-mo	0x0819	2073
Sanskrit	sa	0x044F	1103
Serbian - Cyrillic	sr-sp	0x0C1A	3098
Serbian ?Latin	sr-sp	0x081A	2074
Setsuana	tn	0x0432	1074
Slovenian	sl	0x0424	1060
Slovak	sk	0x041B	1051
Sorbian	sb	0x042E	1070

Spanish - Spain	es-es	0x0C0A	1034
Spanish - Argentina	es-ar	0x2C0A	11274
Spanish - Bolivia	es-bo	0x400A	16394
Spanish - Chile	es-cl	0x340A	13322
Spanish - Colombia	es-co	0x240A	9226
Spanish - Costa Rica	es-cr	0x140A	5130
Spanish - Dominican Republic	es-do	0x1C0A	7178
Spanish - Ecuador	es-ec	0x300A	12298
Spanish - Guatemala	es-gt	0x100A	4106
Spanish - Honduras	es-hn	0x480A	18442
Spanish - Mexico	es-mx	0x080A	2058
Spanish - Nicaragua	es-ni	0x4C0A	19466
Spanish - Panama	es-pa	0x180A	6154
Spanish - Peru	es-pe	0x280A	10250
Spanish - Puerto Rico	es-pr	0x500A	20490
Spanish - Paraguay	es-py	0x3C0A	15370
Spanish - El Salvador	es-sv	0x440A	17418
Spanish - Uruguay	es-uy	0x380A	14346
Spanish - Venezuela	es-ve	0x200A	8202
Sutu	sx	0x0430	1072
Swahili	sw	0x0441	1089
Swedish - Sweden	sv-se	0x041D	1053
Swedish - Finland	sv-fi	0x081D	2077
Tamil	ta	0x0449	1097
Tatar	tt	0X0444	1092
Thai	th	0x041E	1054
Turkish	tr	0x041F	1055
Tsonga	ts	0x0431	1073
Ukrainian	uk	0x0422	1058
Urdu	ur	0x0420	1056
Uzbek ?Cyrillic	uz-uz	0x0843	2115
Uzbek ?Latin	uz-uz	0x0443	1091

Vietnamese	vi	0x042A	1066
Xhosa	xh	0x0434	1076
Yiddish	yi	0x043D	1085
Zulu	zu	0x0435	1077

VBScript 函数参考手册

VBScript TypeName 函数

VBScript 函数参考手册

定义和用法

TypeName 函数可指定变量的子类型。

TypeName 函数可返回的值：

值	描述
Byte	字节值
Integer	整型值
Long	长整型值
Single	单精度浮点值
Double	双精度浮点值
Currency	货币值
Decimal	十进制值
Date	日期或时间值
String	字符串值
Boolean	Boolean 值；True 或 False
Empty	未初始化
Null	无有效数据
<object type>	实际对象类型名
Object	一般对象
Unknown	未知对象类型
Nothing	还未引用对象实例的对象变量
Error	错误

语法

```
TypeName(varname)
```

参数	描述
varname	必需的。变量的名称。

实例

```
dim x
x="Hello World!"
document.write(TypeName(x))
x=4
document.write(TypeName(x))
x=4.675
document.write(TypeName(x))
x=Null
document.write(TypeName(x))
x=Empty
document.write(TypeName(x))
x=True
document.write(TypeName(x))
```

输出：

```
String
Integer
Double
Null
Empty
Boolean
```

VBScript 函数参考手册

VBScript VarType 函数

VBScript 函数参考手册

定义和用法

VarType 函数可返回指示指定变量的子类型的值。

VarType 函数可返回的值：

常数	值	描述
vbEmpty	0	未初始化（默认）
vbNull	1	不包含任何有效数据
vbInteger	2	整型子类型
vbLong	3	长整型子类型
vbSingle	4	单精度子类型
vbDouble	5	双精度子类型
vbCurrency	6	货币子类型
vbDate	7	日期或时间值
vbString	8	字符串值
vbObject	9	字符串子类型
vbError	10	错误子类型
vbBoolean	11	Boolean 子类型
vbVariant	12	Variant （仅用于变量数组）
vbDataObject	13	数据访问对象
vbDecimal	14	十进制子类型
vbByte	17	字节子类型
vbArray	8192	数组

注释：这些常数是由 VBScript 指定的。所以，这些名称可在代码中随处使用，以代替实际值。

注释：假如变量是数组，则 VarType() 会返回 8192 + VarType(数组元素)。举例：整数数组的 VarType() 会返回 8192 + 2 = 8194 。

语法

```
VarType(varname)
```

参数	描述
varname	必需的。变量的名称。

实例

```
dim x
x="Hello World!"
document.write(VarType(x))
x=4
document.write(VarType(x))
x=4.675
document.write(VarType(x))
x=Null
document.write(VarType(x))
x=Empty
document.write(VarType(x))
x=True
document.write(VarType(x))
```

分别输出：

```
String
Integer
Double
Null
Empty
Boolean
```

VBScript 函数参考手册

WCF教程

WCF是Windows通信基础（Windows Communication Foundation）的缩写。 WCF的基本特征是互操作性。这是微软用于构建面向服务的应用程序的最新技术之一。根据基于消息的通信的概念中，一个HTTP请求可以被均匀地表示，WCF是一个统一的API而不管不同的传输机制。

WCF在2006年第一次作为.NET框架以及Windows Vista的一部分发布，然后得到了多次更新。 WCF4.5是当前广泛使用的最新版本。 WCF应用程序由三部分组成 - WCF服务，WCF服务主机和WCF服务客户端。 WCF平台有时也被称为服务模型。

WCF的基本概念

消息 - 这是由几部分组成身外的通信单元。消息实例被作为接收的所有类型的客户端和服务之间的通信。

端点 - 它定义了一个消息将被发送或接收的地址。它还指定的通信机制来描述如何将邮件将被界定的组消息一起发送。一个端点的结构包括以下几个部分组成。

- 地址 - 该指定要接收的消息的准确位置和被指定为一个统一资源标识符（URI）。它表现为方式：//域名[：端口]/[路径]。这可通过具有上述的地址一目了然很好理解。

  net.tcp://localhost:9000/ServiceA

  在这里，net.tcp是TCP协议方案。该域名是本地主机，可一机多用的名称或网络域和路径是ServiceA。

- 绑定 - 它定义了一个端点通信，并包括一些使通信基础设施的结合元件的方式。例如，结合状态用于运输如TCP，HTTP等，消息编码和相关的安全性以及可靠性的协议格式的协议。

- 合约 - 这是一个范围的操作来指定该消息的端点将通信。它通常是一个接口名称

主机 - 主机相对于WCF通常意味着WCF服务的主机可以通过许多可用的选项，如自托管完成，IIS托管和主持。

元数据 - 这是WCF的一个显著的概念，因为它方便了客户端应用程序和一个WCF服务之间的轻松互动。通常，元数据的WCF服务被启用时，自动生成的，这是由服务和它的端点的检查完成。

WCF客户端 - 被揭露的服务操作的方法形式被称为WCF客户端创建客户端应用程序。这可以由任何应用程序，即使是在一个没有服务的主机托管。

通道 - 通道是通过该客户端可以与服务进行通信的介质。许多不同类型的通道可以得到叠层和被称为信道栈。

SOAP - 虽然名为“简单对象访问协议”，SOAP不是一个传输协议，取而代之的是一个XML文档，其包括头部和主体部。

WCF的优点

1.它是可互操作相对于其他的服务。与此形成鲜明对比.NET远程处理，客户端和服务必须有.NET。

2. WCF服务的提供增强的可靠性和安全性相比，ASMX（活动服务器的方法）web服务。

3.实现安全模型，并结合不断变化的WCF不需要编码的重大变化。只需很少的配置变更，必须符合的约束。

4. WCF具有内置在记录机制，而在其他技术中，必须做必要的编码。

5. WCF集成AJAX和支持JSON（JavaScript对象表示法）。

6.提供可扩展性可支持出新的Web服务标准。

7.具有极其强大的默认安全机制。

WCF与Web服务/Web Service - WCF教程

下面列出了WCF和Web服务之间存在一些重大差异。

- 属性：WCF服务是通过定义ServiceContract和OperationContract属性，而在Web服务，WebService和WebMethod属性用于定义相同。
- 协议：WCF支持多种协议，即HTTP，命名管道，TCP和MSMQ；而Web服务仅支持HTTP协议。
- 托管机制：WCF托管不同的激活机制，即IIS（Internet信息服务），WAS（Windows激活服务），自托管和Windows服务，而Web服务则只能由IIS托管。
- 服务：WCF支持一个强大的安全，值得信赖的消息传递，事务性和互操作性，而Web服务只支持保障服务。
- 序列化：WCF支持DataContract串行采用System.Runtime.Serialization，而Web服务通过使用System.Xml.Serialization支持XML序列化。
- 工具：ServiceMetadata工具（svcutil.exe）用于客户机生成的WCF服务而WSDL.EXE工具用来产生相同web服务。
- 异常处理：在WCF中，未处理的异常都是在一个更好的方式通过使用FaultContract处理，并没有得到Web服务SOAP（简单对象访问协议）故障返回给客户端等。
- 有可能要序列哈希Tablein WCF，但这不能在web服务中。
- 绑定：WCF支持多种类型，如 basicHttpBinding，WSDualHttpBinding，WSHttpBinding等绑定，而Web服务仅支持SOAP或XML。
- 多线程：WCF支持多线程利用ServiceBehavIor类，而这Web服务不支持。
- 双工服务操作：WCF支持双工服务业务除了支持单向和请求 - 响应服务操作，而Web服务不支持双工服务操作。

WCF开发工具 - WCF教程

开发一个WCF服务应用程序，主要有两种工具 - Microsoft Visual Studio和Code Plex。Microsoft Visual Studio是一个完整的包所必需的开发了大量的象ASP.NET Web应用，桌面应用，移动应用和许多不同的应用程序的开发工具。 .NET框架的功能，采用的是微软的Visual Studio。Code Plex另一方面是微软的开源项目托管网站，提供一些免费的工具，WCF服务应用程序的开发。

Microsoft Visual Studio

有许多Microsoft Visual Studio版本，最初，它(Visual Studio 2005)不是WCF开发的热情支持者。目前，Visual Studio 2008是唯一的Microsoft IDE提供WCF服务应用程序的开发。如今，微软Visual Studio 2010最新版本，也是开发WCF服务应用程序的首选工具。更重要的是，现在在Visual Studio中没有开发WCF服务应用程序中的现成模板。

选择这样的模板的引出另外用于下列目的的文件 -

- 服务合约
- Service实现
- 服务配置

微软Visual Studio创建了一个简单的“Hello World”的服务时都会自动添加一些必要的属性，甚至不用编写任何代码。

Code Plex

CodePlex网站是由微软于2006年6月推出，从那时起就一直是由大量世界各地的开发人员用于创建.NET项目而使用。一些由CodePlex上开发WCF服务应用程序所提供的工具以下。

- wscf.blue - Microsoft Visual加载项和“协定优先”的开发工具集简化定义WCF服务操作，并相应产生一个代码框架。

  对于相同的一个重要环节是 http://wscfblue.codeplex.com/

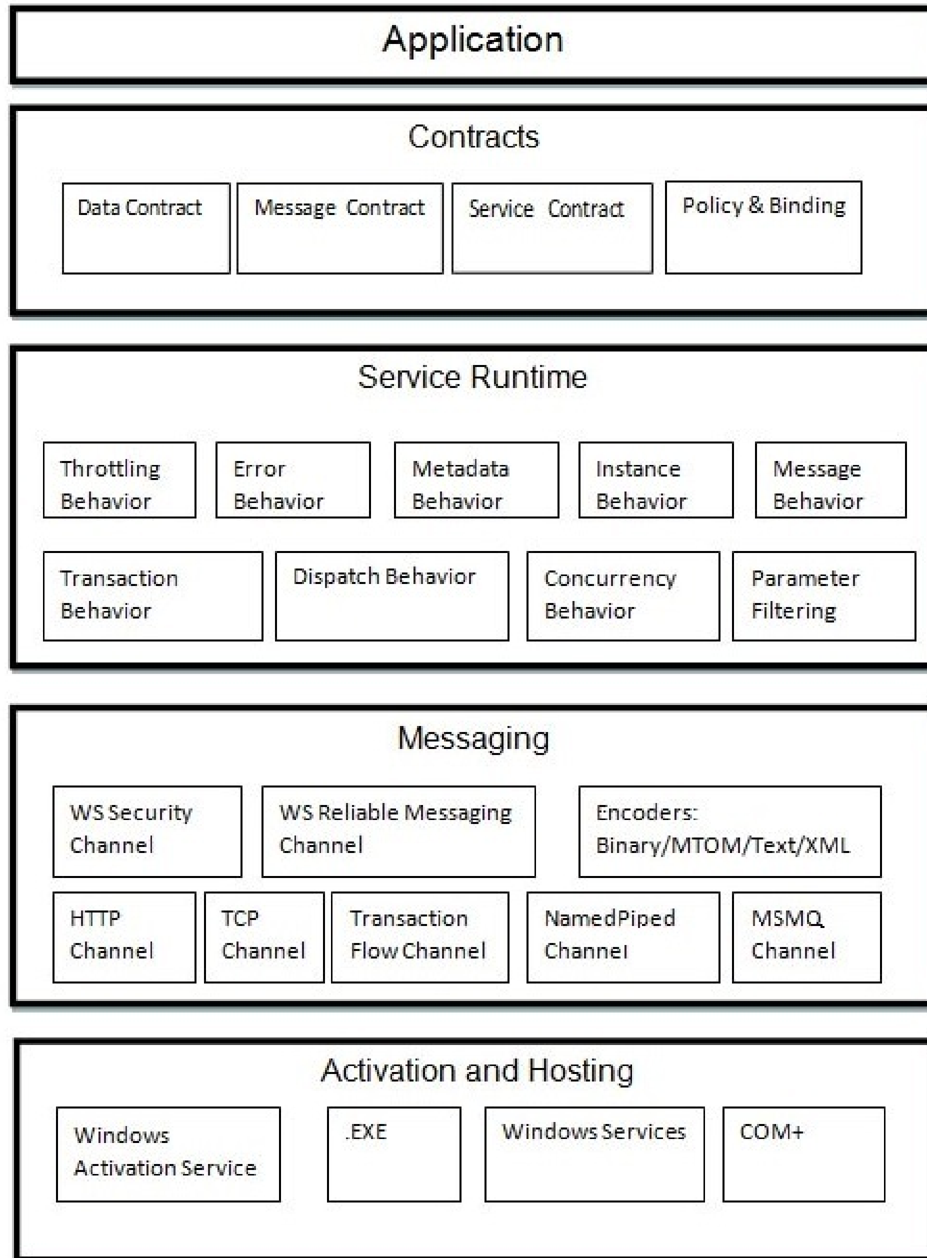
- WCFProxyGenerator - 这也是微软的Visual Studio插件。该工具被用来扩展客户端生成并提供额外的错误处理。对于有关于这个特定的开发工具，更多信息请访问 http://wcfproxygenerator.codeplex.com.

- WCFMock - WCF服务的测试是一个复杂的任务，这个开发工具提供了WCF服务通过它的有用的类单元测试的一个方便的解决方案。有关此工具，更多信息请访问 http://wcfmock.codeplex.com/.

另一种WCF服务应用程序开发免费工具是 WCFStorm。它的Lite版本提供了许多显着的特征来动态调用和测试WCF服务，编辑服务绑定，修改WCF URL端点等等。

WCF架构 - WCF教程

WCF是一个分层架构，为开发各种分布式应用用的充分支持。该体系结构在下面将详细说明。



约定

约定层旁边就是应用层，并含有类似于现实世界的约定，指定服务和什么样的信息可以访问它会使操作的信息。约定基本都是在简短的讨论如下四种类型。

- **Service contract** - 约定规定，在沟通过程中使用的信息给客户端，以及对终端的产品和协议的外部世界。

- **Data contract** - 由服务交换的数据是由一个数据契约定义。客户端和服务需要在与数据合同协议。
- **Message Contract** - 数据合同由约定信息控制。它主要是SOAP消息的参数类型格式的定制。在此，应该提到的是WCF采用SOAP格式进行通信。 SOAP代表简单对象访问协议。
- **Policy and Binding** - 由策略和有约束力的约定被定义为一个服务，这样的条件下通信的某些先决条件。客户端需要遵循这一约定。

服务运行时

服务运行时层仅仅是约定层之下。它指定在运行时出现的各种服务行为。有许多类型的行为，可以进行配置，如下的服务运行。

- **Throttling Behavior** - 管理处理的消息的数量
- **Error Behavior** – 定义任何内部服务错误发生的结果
- **Metadata Behavior** – 指定的元数据的可用性到外界
- **Instance Behavior** – 定义要创建需要的实例的数量，以使它们可用于在客户端
- **Transaction Behavior** – 能够在事务状态的变化的情况下出现任何故障
- **Dispatch Behavior** - 控制由该消息得到了WCF的基础处理方式
- **Concurrency Behavior** - 控制的客户端 - 服务器通信过程中并行运行功能
- **Parameter Filtering** - 功能参数的方法验证的过程，在它被调用之前

消息

这层几个通道构成主要涉及两个端点之间传送的消息的内容。一组通道的形成通道堆栈和构成的通道堆栈的下面那些渠道的两种主要类型。

- **Transport Channels** - 这些通道都存在于栈底和负责发送和接收使用像HTTP，TCP，P2P，命名管道和MSMQ传输协议的消息。
- **Protocol Channels** - 存在于堆栈的顶部，这些信道也被称为层状通道通过修改消息实现线级协议。

激活和托管

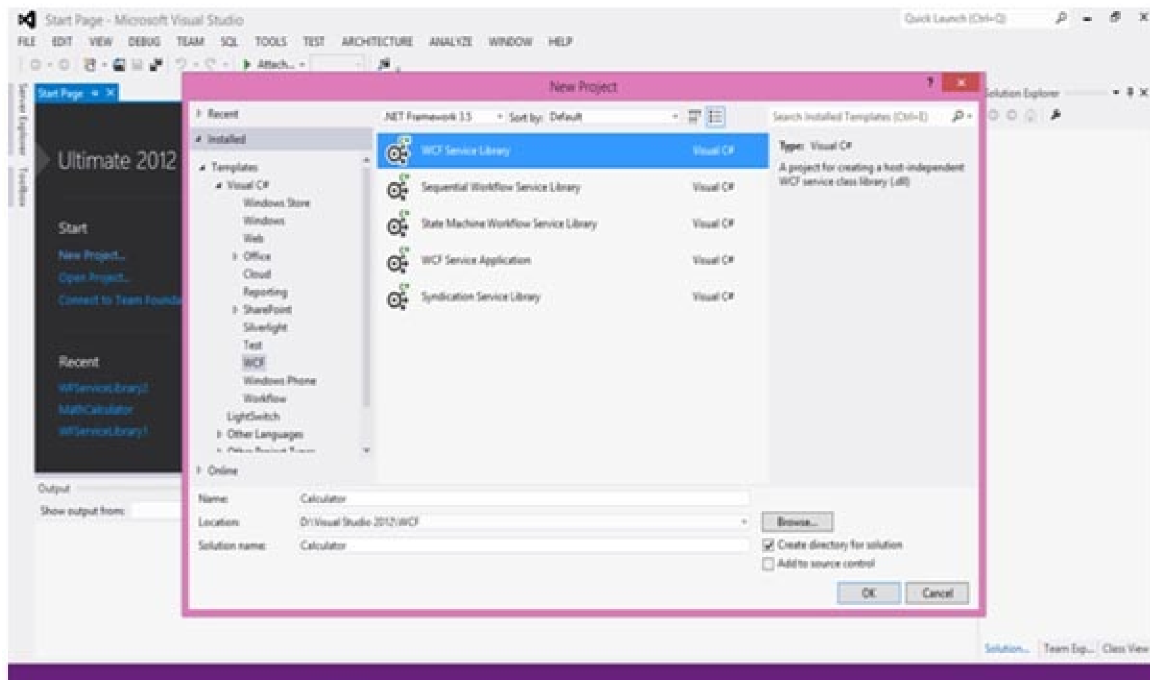
WCF的体系结构的最后一层是其中服务实际上是托管，或者可以以方便客户端被执行的地方。这是通过在下面简要讨论的各种机制进行。

- **IIS** - 互联网信息服务的缩写提供使用HTTP协议通过服务优势很多。这里主机代码的要求不是强制性的，用于激活该服务代码，相反，服务码被自动激活。
- **Windows**激活服务 - 这就是俗称WAS和带有IIS7.0。 HTTP和非HTTP通信，可以在这里通过使用TCP或Namedpipe协议。
- **Self-Hosting** - 这是由一个WCF服务获取自托管的控制台应用程序的机制。这种机制提供了惊人的灵活选择所需的协议和设置自己的解决方案方面。
- **Windows Service** - 主持这一机制的WCF服务是有利的，因为该服务保持激活状态，并接触到客户端，由于没有运行时激活。

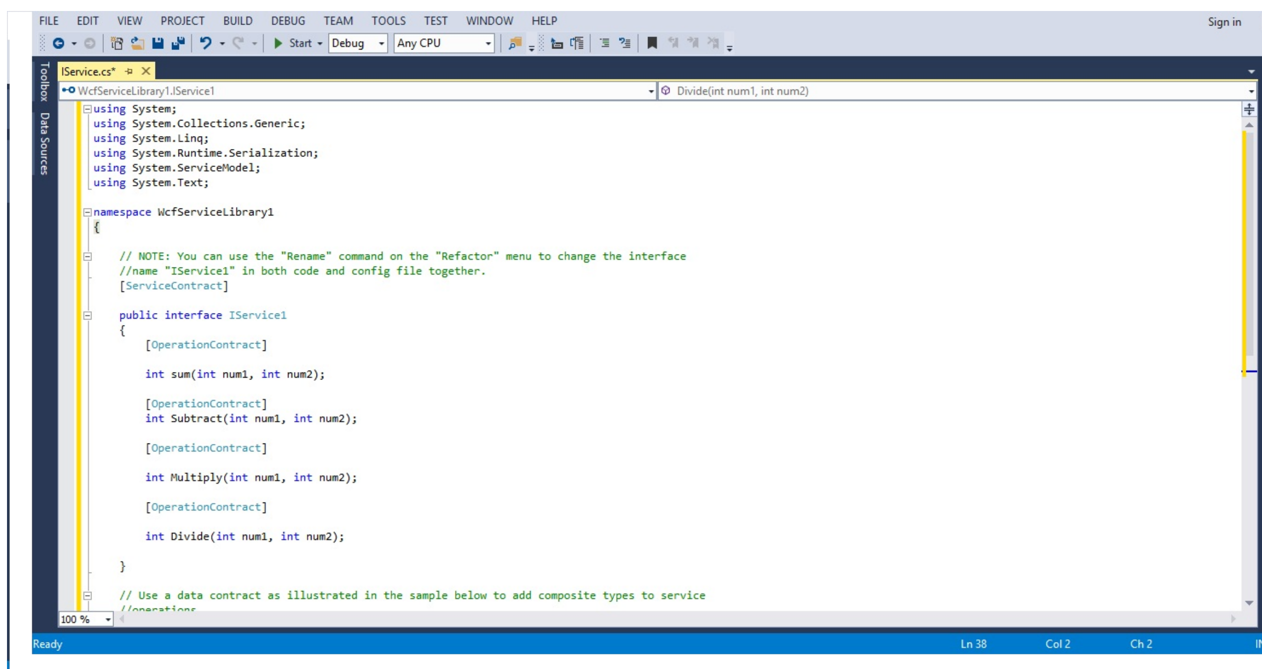
创建WCF服务 - WCF教程

使用Microsoft Visual Studio2012创建WCF服务，理解如下所有必要的编码，更好地创建WCF服务的概念，这里做一个简单的任务。

- 启动Visual Studio 2012。
- 单击新建项目，然后在Visual C#标签，选择WCF选项。



WCF服务创建，执行如加法，减法，乘法和除法基本的算术运算。主要的代码是在两个不同的文件 - 一个接口和一个类。



一个WCF中包含一个或多个接口和实现类。

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace WcfServiceLibrary1
{
   //NOTE: You can use the "Rename" command on the "Refactor" menu to change the
   //interface name "IService1" in both code and config file together.
   [ServiceContract]
   Public interface IService1
   {
      [OperationContract]
      int sum(int num1, int num2);

      [OperationContract]
      int Subtract(int num1, int num2);

      [OperationContract]
      int Multiply(int num1, int num2);

      [OperationContract]
      int Divide(int num1, int num2);
   }

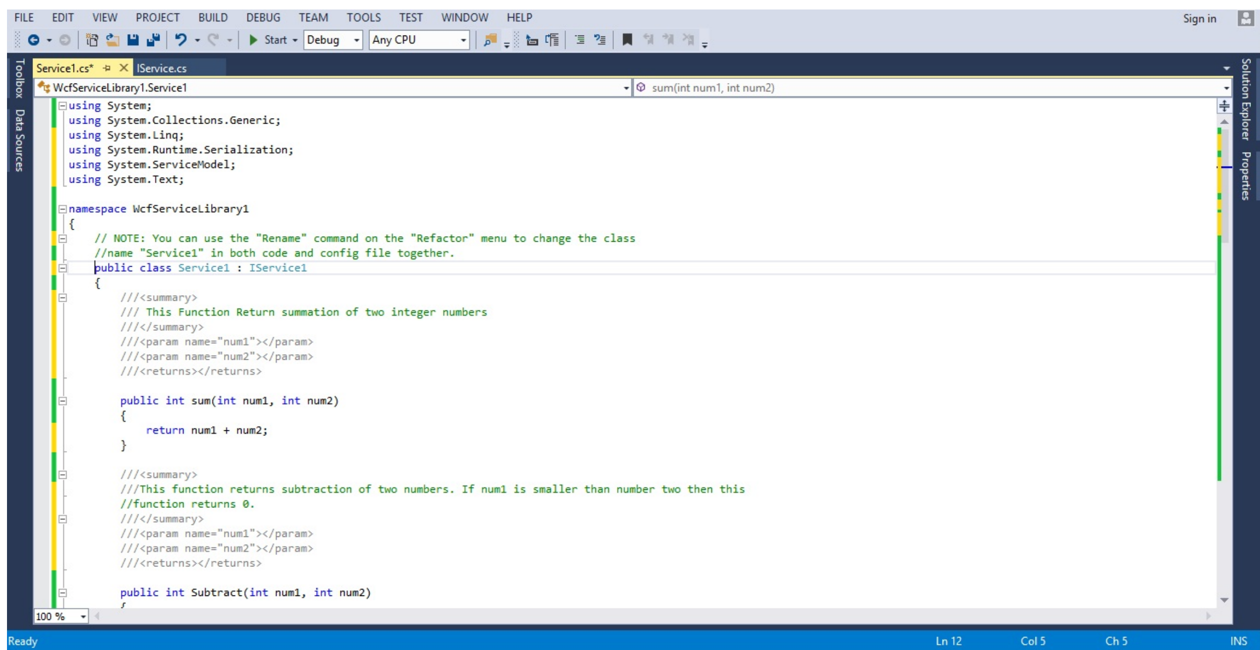
   //Use a data contract as illustrated in the sample below to add composite types
   //to service operations.
   [DataContract]
   Public class CompositeType
   {
      Bool boolValue = true;
      String stringValue = "Hello ";

      [DataMember]
      Public bool BoolValue
      {
         get { return boolValue; }
         set { boolValue = value; }
      }

      [DataMember]

      Public string StringValue
      {
        get { return stringValue; }
        set { stringValue = value; }
      }
   }
}
```

而其后面是类的代码，


FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Sign in
Start
Debug
Any CPU
Service1.cs*
IService.cs
WcfServiceLibrary1.Service1
sum(int num1, int num2)
Toolbox
Data Sources
Solution Explorer
Properties
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;
namespace WcfServiceLibrary1
{
// NOTE: You can use the "Rename" command on the "Refactor" menu to change the class
//name "Service1" in both code and config file together.
public class Service1 : IService1
{
///<summary>
/// This Function Return summation of two integer numbers
///</summary>
///<param name="num1"></param>
///<param name="num2"></param>
///<returns></returns>
public int sum(int num1, int num2)
{
return num1 + num2;
}
///<summary>
///This function returns subtraction of two numbers. If num1 is smaller than number two then this
//function returns 0.
///</summary>
///<param name="num1"></param>
///<param name="num2"></param>
///<returns></returns>
public int Subtract(int num1, int num2)
100 %
Ready
Ln 12
Col 5
Ch 5
INS

```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Runtime.Serialization;
usingSystem.ServiceModel;
usingSystem.Text;

namespace WcfServiceLibrary1
{
   //NOTE: You can use the "Rename" command on the "Refactor" menu to change the
   //class name "Service1" in both code and config file together.

   publicclassService1 :IService1
   {
      /// This Function Return summation of two integer numbers

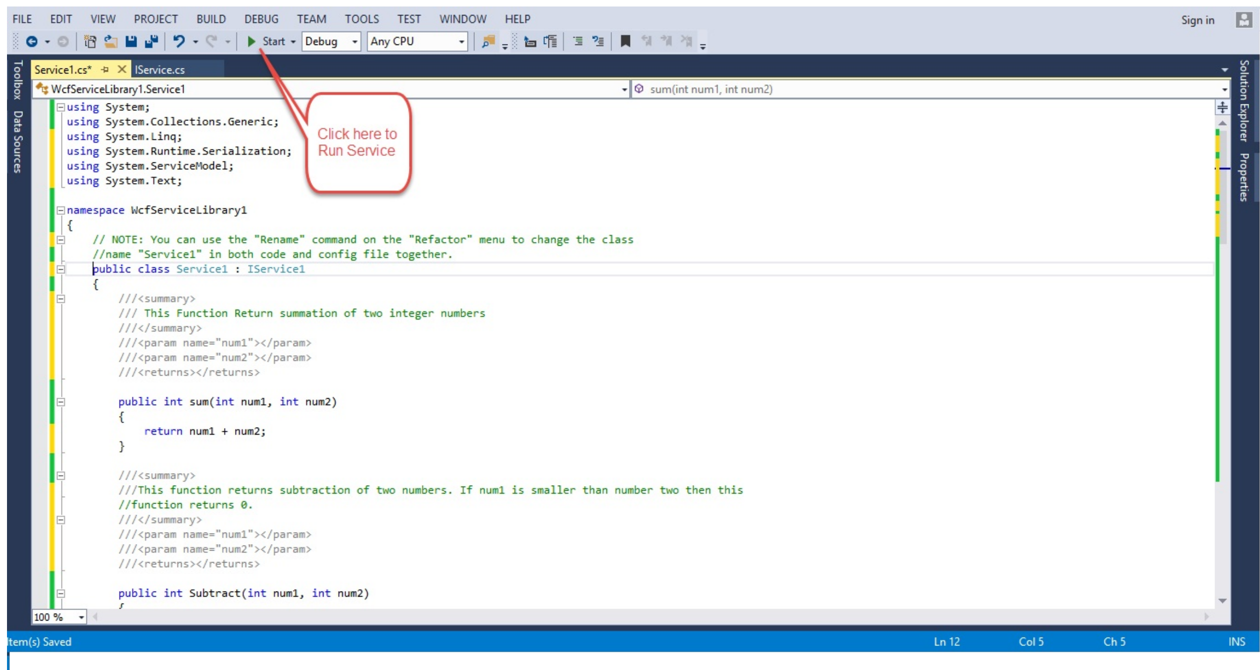
      publicint sum(int num1, int num2)
      {
         return num1 + num2;
      }
      ///This function returns subtraction of two numbers.
      ///If num1 is smaller than number two then this function returns 0.

      publicint Subtract(int num1, int num2)
      {
         if (num1 > num2)
         {
            return num1 - num2;
         }
         else
         {
            return 0;
         }
      }
      ///This function returns multiplication of two integer numbers.

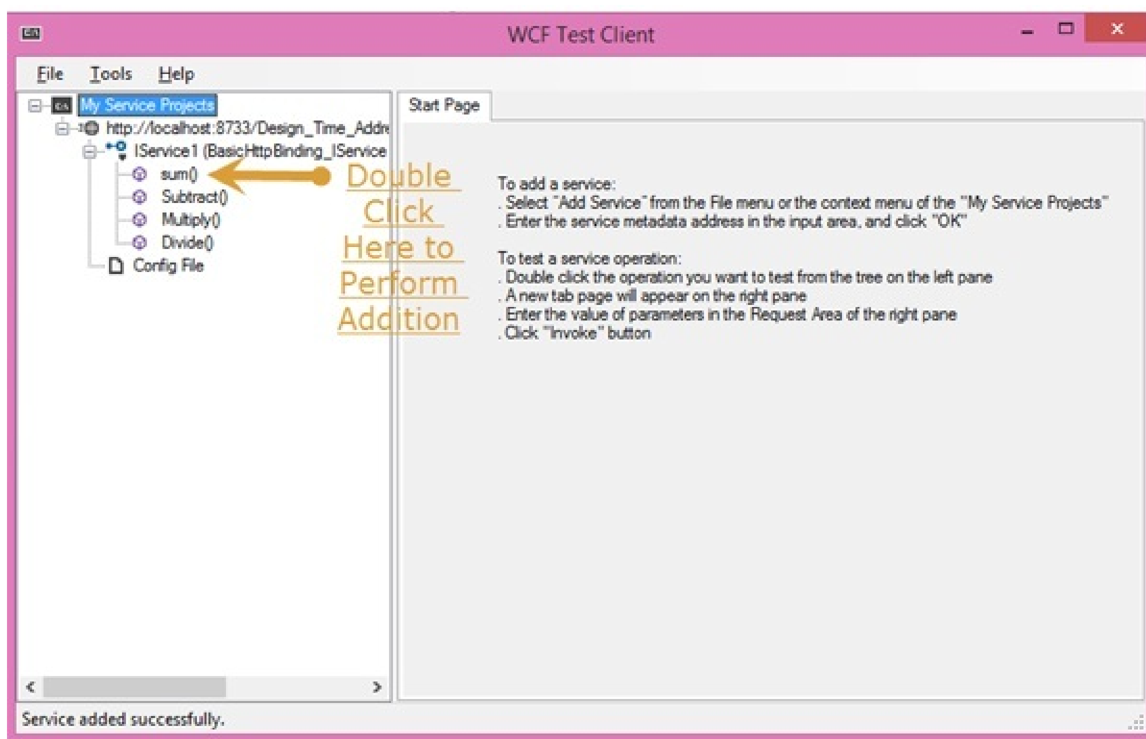
      publicint Multiply(int num1, int num2)
      {
         return num1 * num2;
      }
      ///This function returns integer value of two integer number.
      ///If num2 is 0 then this function returns 1.

      publicintDivide(int num1, int num2)
      {
         if (num2 != 0)
         {
            return (num1 / num2);
         }
         else
         {
            return 1;
         }
      }
   }
}
```

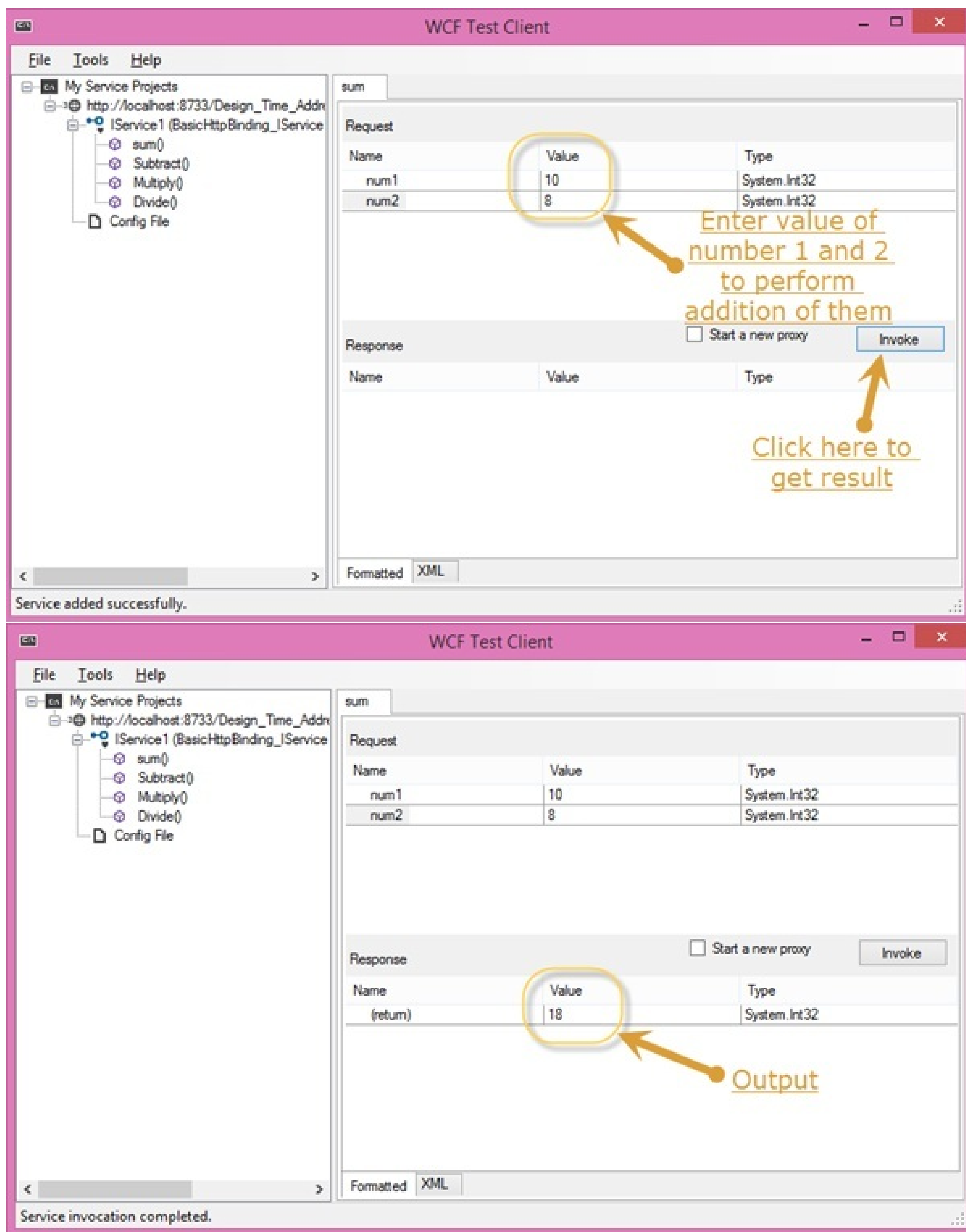
要运行此服务，请在Visual Studio中点击开始按钮。



当我们运行这个服务，下面的屏幕会出现。

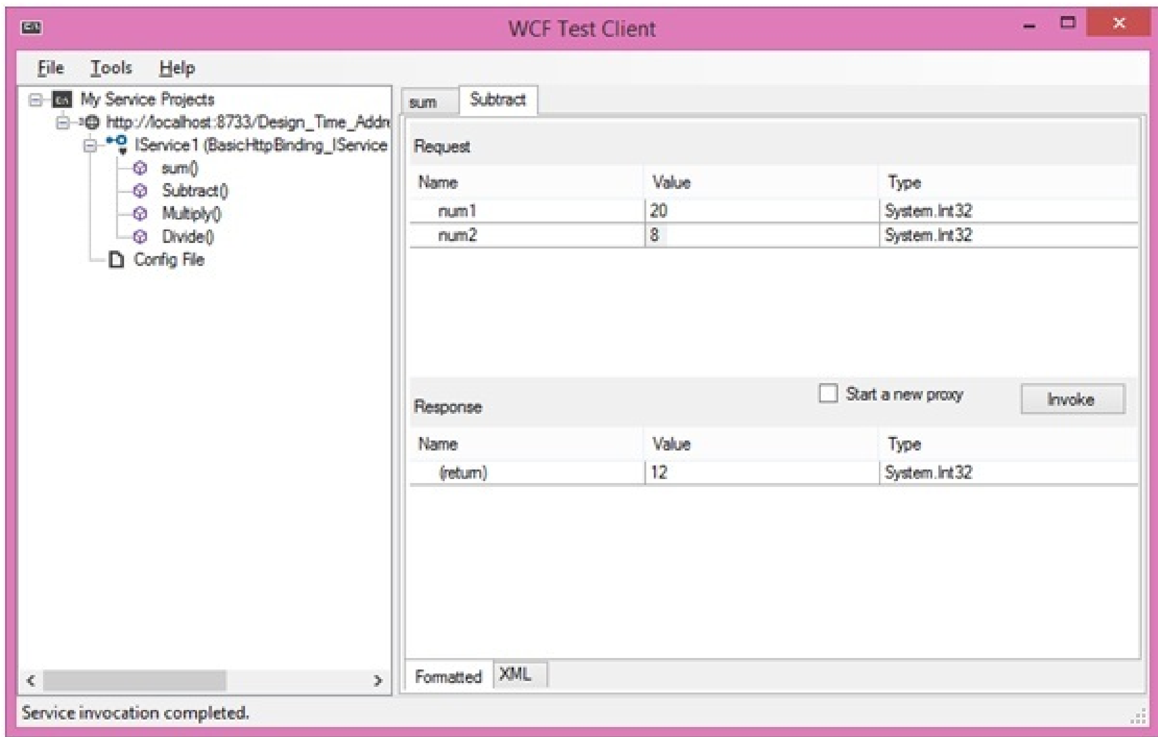


上点击sum方法，在下面的页面将被打开。在这里，可以输入任何两个整数，然后单击Invoke按钮。该服务将返回这两个数字的总和。

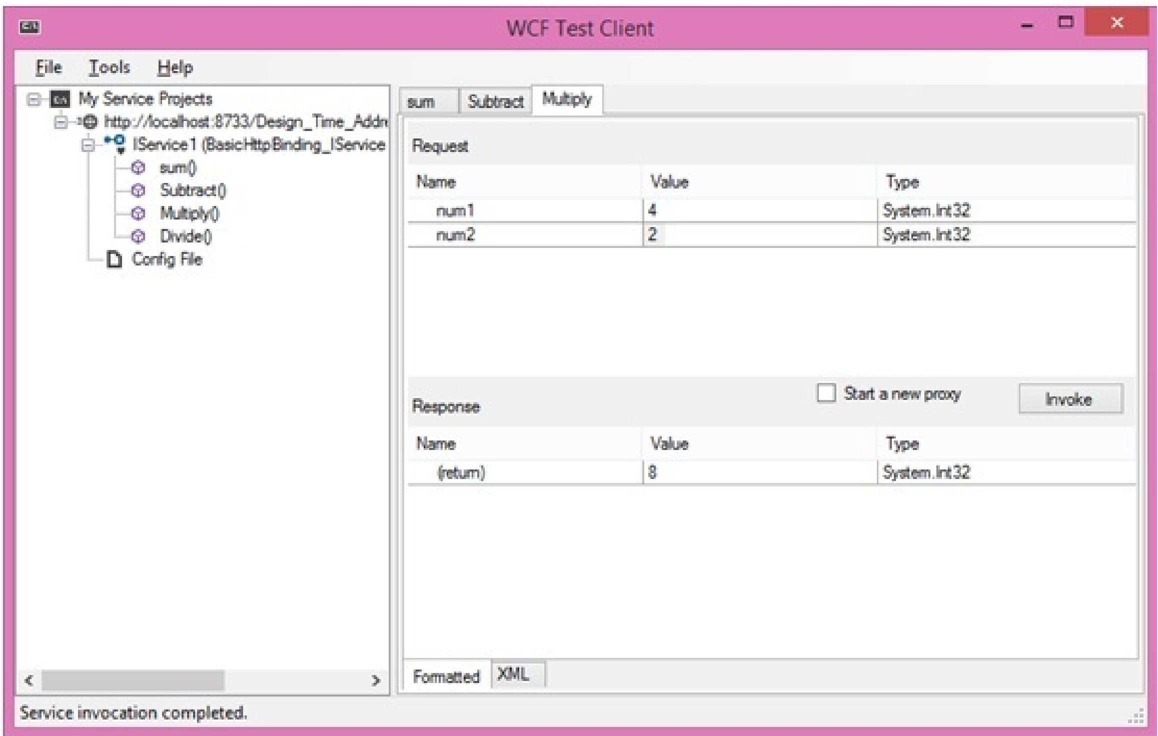


像求和，我们可以执行哪个都列在菜单中的所有算术运算。这里是捕捉他们。

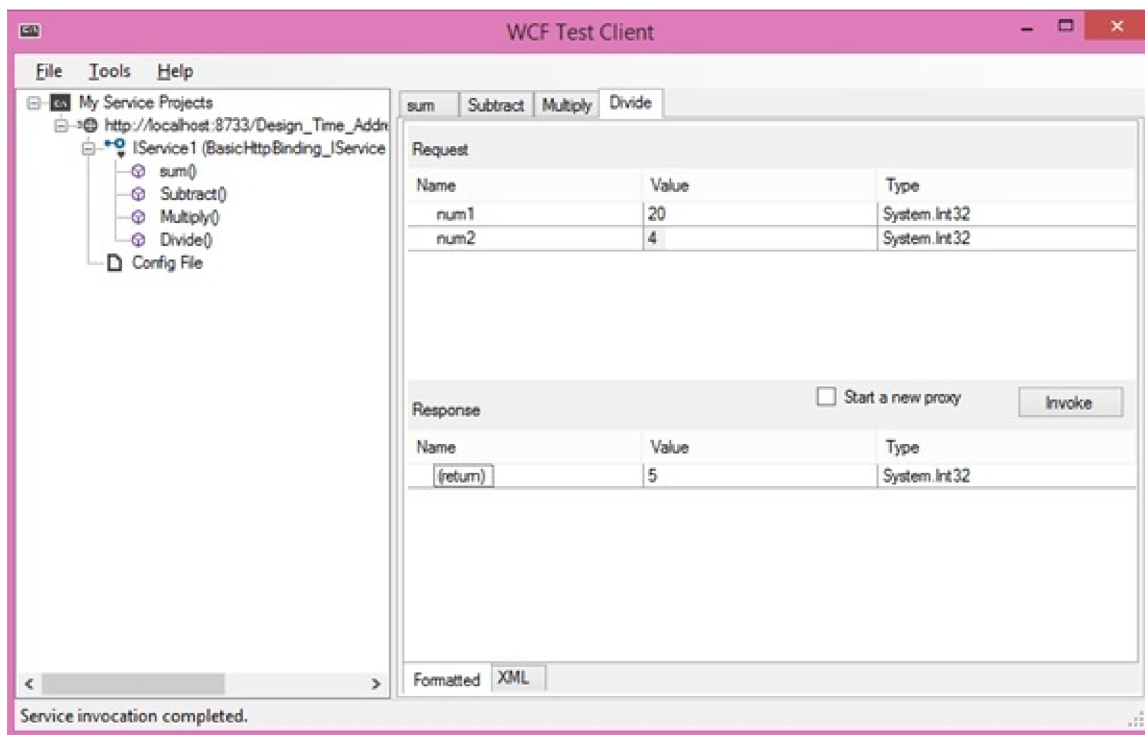
当点击下页将出现在Sutbtarct方法。输入整数，点击调用按钮，得到的输出如下所示。



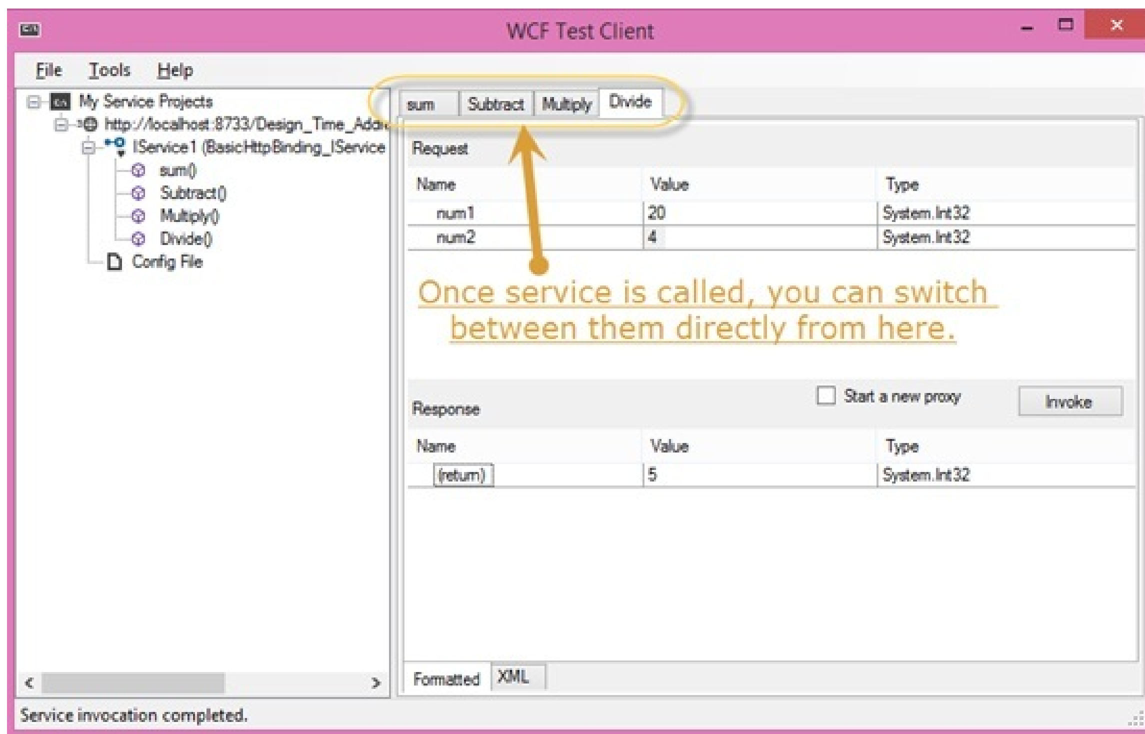
下页将出现在Multiply方法单击时。输入整数，点击调用按钮，得到的输出如下所示。



下面的页面上会出现当点击Divide方法时。输入整数，点击调用按钮，得到的输出如下所示。



一旦服务被调用，可以在它们之间，直接从这里切换。



主机WCF服务 - WCF教程

建立一个WCF服务后，下一步就是托管它，以便客户端应用程序可以使用，这就是所谓的WCF服务托管。 WCF服务可以通过使用任何的四种方法如下托管。

- **IIS**主机 - IIS是Internet信息服务的缩写。它的工作模式是类似于ASP.NET，而托管的WCF服务。IIS托管的最大的特点是服务激活自动处理。 IIS主机还提供过程的健康监测，闲置关机，进程回收，还有更多的功能，以方便WCF服务托管。
- 自助主机 - 在一个WCF服务托管在托管应用程序中，它被称为自主机。它要求开发人员编写必要的编码ServiceHost 初始化。在自托管，WCF服务可以在各种类似控制台应用程序，Windows窗体等应用程序托管
- **WAS**主机 - 在Windows激活服务主机的WCF服务(WAS)，它的功能如进程回收，空闲时间管理，通用配置系统，支持HTTP，TCP等
- **Windows**服务主机 - 本地系统的客户端，这是最好的承载WCF服务作为一个窗口服务，这就是所谓的窗口服务主机。所有的Windows版本支持这种类型的托管，服务控制管理器可以控制WCF服务的流程生命周期。

消费WCF服务 - WCF教程

Windows通讯基础(WCF)服务允许其他应用程序访问或使用它们。 WCF服务可以消费，或由根据主机类型的方式访问。这里，我们说明由步骤方法的步骤以消费WCF服务，每个受欢迎的托管选项，即 –

- 消费WCF服务托管在IIS5/6
- 自托管消费WCF服务
- 消费WCF服务托管在Windows激活服务
- 消费WCF服务托管在Windows服务

WCF服务绑定 - WCF教程

WCF服务绑定是一个集合，每个元素定义了服务与客户端进行通信方式的几个元素。传输元素和一个消息编码元素各自结合两个最重要的组成部分。这里是WCF服务绑定常用的列表。

基础绑定

基础约束是由basicHttpBinding的类提供的，这种结合使用HTTP协议进行传输为目的，并代表一个WCF服务作为一个ASP.NET Web服务(ASMX Web服务)，这样方便ASMX Web服务的老客户可以使用新服务。这被设置为默认的受Silverlight启用WCF Web服务绑定，是一个标准Web服务通信的风格结合。这并不支持可靠的消息。

在下文中介绍的代码片段，描绘的默认设置基础绑定。

```
<basicHttpBinding>
  <binding name="basicHttpBindingDefaults" allowCookies="false"
     bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"
     maxBufferPoolSize="524288" maxBufferSize="65536" maxReceivedMessageSize="65536"
     messageEncoding="Text" proxyAddress="" textEncoding="utf-8" transferMode="Buffer"
     useDefaultWebProxy="true" closeTimeout="00:01:00" openTimeout="00:01:00"
     receiveTimeout="00:10:00" sendTimeout="00:01:00">
        <readerQuotas maxArrayLength="16384" maxBytesPerRead="4096" maxDepth="32"
           maxNameTableCharCount="16384" maxStringContentLength="8192"/>
        <security mode="None">
           <transport clientCredentialType="None" proxyCredentialType="None" realm=""/>
           <message algorithmSuite="Basic256" clientCredentialType="UserName" />
        </security>
  </binding>
</basicHttpBinding>
```

上面的默认设置有其明显的局限性邮件大小是有限的，在这里安全模式也无法比拟。但是基本的结合解决了这个问题类似下面的定制。

```
<basicHttpBinding>
  <binding name="basicHttpSecure" maxBufferSize="100000"
     maxReceivedMessageSize="100000">
     <readerQuotas maxArrayLength="100000" maxStringContentLength="100000"/>
     <security mode="TransportWithMessageCredential" />
  </binding>
</basicHttpBinding>
```

Web服务（WS）绑定

这是通过WSHttpBinding类提供，此绑定相似于基础约束，并使用相同的协议进行传输，但提供了几个WS- 规范，比如*WS-* 可靠消息，*WS-* 事务，*WS-* 安全，还有更多。简而言之，*WsHttpBinding*等于总结*basicHttpBinding*和*WS-* 规范。在这里，在下文中介绍的代码片段，

说明默认设置WS绑定。

```
<wsHttpBinding>
  <binding name="wsHttpBindingDefaults" allowCookies="false" bypassProxyOnLocal="false"
     closeTimeout="00:01:00" hostNameComparisonMode="StrongWildcard"
     maxBufferPoolSize="524288" maxReceivedMessageSize="65536" messageEncoding="Text"
     openTimeout="00:01:00" receiveTimeout="00:10:00" proxyAddress=""
     sendTimeout="00:01:00" textEncoding="utf-8" transactionFlow="false"
     useDefaultWebProxy="true" >
        <readerQuotas maxArrayLength="16384" maxBytesPerRead="4096" maxDepth="32"
           maxNameTableCharCount="16384" maxStringContentLength="8192"/>
        <reliableSession enabled="false" ordered="true" inactivityTimeout="oo:10:00" />
        <security mode="Message">
           <message algorithmSuite="Basic256" clientCredentialType="Windows"
              esatalishSecurityContext="true" negotiateServiceCredential="true" />
           <transport clientCredentialType="Windows" proxyCredentialType="None" realm=""/
        </security>
  </binding>
</wsHttpBinding>
```

IPC绑定

这种结合使得使用命名管道，由netNamedPipeBinding类提供。这是最快的约束和所有可用的绑定是最安全的。虽然，消息级安全性这里不支持，消息是因为一个强大的运输保障的默认安全。在这里，下面的代码片段，说明默认设置为IPC结合。

```
<netNamedPipeBinding>
  <binding name="netPipeDefaults" closeTimeout="00:01:00"
     hostNameComparisonMode="StrongWildcard" maxBufferPoolSize="524288"
     maxBufferSize="65536" maxConnections="10" maxReceivedMessageSize="65536"
     openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
     transactionFlow="false" transactionProtocol="OleTransactions"
     transferMode="Buffered" >
        <readerQuotas maxArrayLength="16384" maxBytesPerRead="4096" maxDepth="32"
           maxNameTableCharCount="16384" maxStringContentLength="8192"/>
        <security mode="Transport">
        </security>
  </binding>
</netNamedPipeBinding>
```

其他类型的服务绑定如下：

- TCP Binding - 由NetTcpBinding类结合TCP协议的通信在同一网络内，并且不会以二进制格式信息编码。这种结合被认为是最可靠的对比。
- WS Dual Binding - 这种结合便于双向通信，即消息可以被发送和接收的客户端和服务的唯一例外的是wsHttpBinding。这是由WSDualHttpBinding类提供的。
- Web binding - 这种结合被设计为表示WCF服务中的HTTP请求的形式，通过使用HTTP的GET和HTTP的POST等方式，这是可用的WebHttpBinding类，并与社会网络常用。

- MSMQ Binding - 这个绑定由NetMsmqBinding类，还提供用于提供在情况下，服务于一个不同于客户端发送的处理消息时间的解决方案。这种结合使得使用MSMQ传输，并提供支持的消息队列。 MSMQ是微软提供的队列消息实现。
- Federated WS Binding - 这种结合是由WSFederationHttpBinding类提供。这是WS结合的一种具体形式，并提供支持，以联合安全。
- Peer Network Binding - 由NetPeerTCPBinding类提供，该结合主要是用在文件共享系统，例如种子和TCP协议中使用。它使用TCP协议等网络运输。在这个网络中每个机器（节点）充当客户端和一个服务器到另一个节点。这是用在像奔流的文件共享系统。
- MSMQ Integration binding - 这种结合是由MsmqIntegrationBinding类提供的。这种结合提供支持MSMQ（微软消息队列），使现有通信系统进行通信。

除了这些，还可以创建自定义绑定。然而，由于它能够调整每个WCF配置属性绑定，需要创建自定义绑定的产生极少。

WCF实例管理 - WCF教程

这组由Windows通讯基础(WCF)结合一组消息(客户端请求)服务实例所采用的技术被称为实例管理。一个完全由三种类型实例激活支持WCF，它们如下所述。

1.每个调用服务

每次调用服务是Windows通讯基础的默认实例激活模式。当一个WCF服务配置为每个调用服务，一个CLR对象是时间跨度客户调用或请求进行创建。 CLR代表公共语言运行库，并在WCF服务实例。

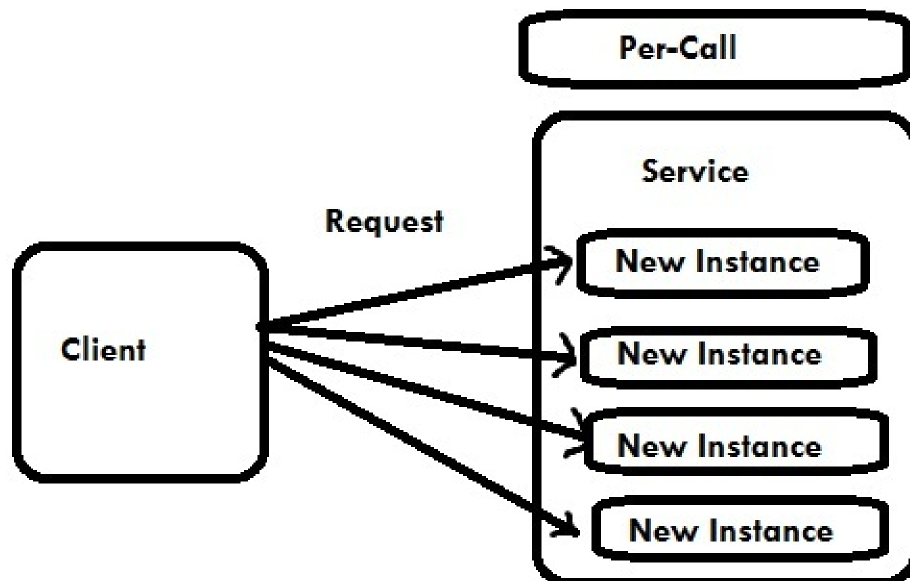
在每个调用服务，每一个客户端请求实现专用消耗相同的内存并且新的服务实例较少，相较于其他类型的实例激活。必需有InstanceContextMode属性，以指示WCF服务以充当每次调用服务被设置为InstanceContextMode.PerCall。InstanceContextMode属性属于ServiceBehavior属性。

因此，每调用服务可以被配置为

```
[ServiceContract]
interface IMyContract
{...}
[ServiceBehavior (InstanceContextMode = InstanceContextMode.PerCall)]
class MyService : IMyContract
{...}
```

服务在这里表示为IMyContract。

每次调用服务实例激活的过程可以描述如下图。



实现每个调用服务

```
[DataContract]
  class Param {....}
  [ServiceContract]
  interface IMyContract
  {
     [OperationContract]
     void MyMethod(Param objectIdentifier);
  }
  class MyPerCallService : IMyContract, IDisposable
  {
     public void MyMethod(Param objectIdentifier)
     {
        GetState(objectIdentifier);
        DoWork();
        SaveState(objectIdentifier);
     }
     void GetState(Param objectIdentifier) {....}
     void DoWork() {....}
     void SaveState(Param objectIdentifier) {....}
     public void Dispose() {....}
  }
```

这里，参数是用于创建对上述实施例的模拟类型的参数。

2.每个调用服务

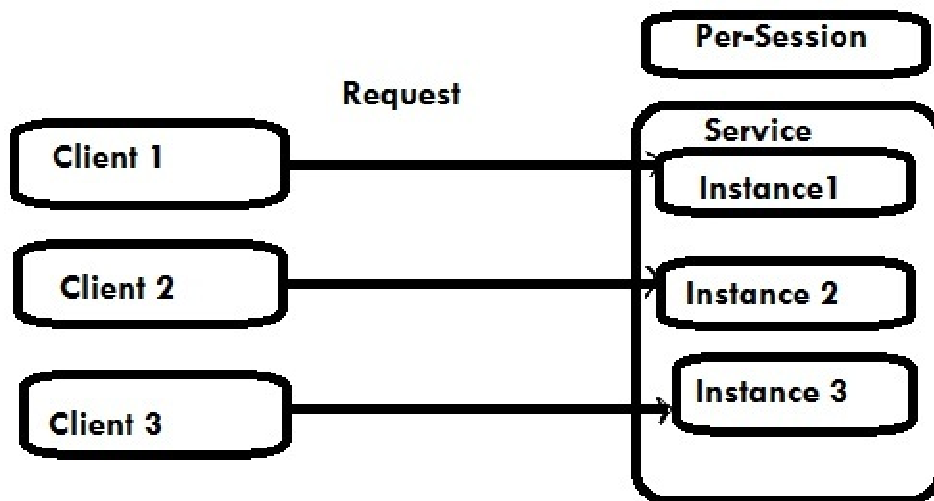
在此激活WCF模式，私有或者我们可以说这是一个保密的会话保持两个实体，即客户端和特定的服务实例。也被称为私有会话服务，该模式提供了其始终致力于为每一个客户要求和自主各有关该会话感知服务的其他情况下的一个新的服务实例。

InstanceContextMode属性需要设置为PerSession发起这个每会话服务。在这里，服务实例保留在内存中全部通过会话持续时间。从可扩展性的激活模式受到所配置的服务是不是能够支持任何额外出色的客户比其他几个或可能达到一些，因为涉及的每一个专用服务实例的成本。

因此，每个会话服务可以被配置为

```
[ServiceBehavior (InstanceContextMode = InstanceContextMode.PerSession)]
class MyService : IMyContract
{...}
```

每个会话服务的过程可以被描述为下面的图。



下面的代码显示了配置为私有会话的使用合约和服务。输出表示该客户端确实有一个专门的服务实例。

服务代码

```
[ServiceContract(Session = true)]
  interface IMyContract
  {
     [OperationContract]
     void MyMethod();
  }
  [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]
  class MyService : IMyContract, IDisposable
  {
     int m_Counter = 0; MyService() { Console.WriteLine("MyService.MyService()"); }
     public void MyMethod()
     {
        m_Counter++;
        Console.WriteLine("Counter = " + m_Counter);
     }
     public void Dispose()
     {
        Console.WriteLine("MyService.Dispose()");
     }
  }
```

客户端代码

```
MyContractProxy proxy = new MyContractProxy(); proxy.MyMethod(); proxy.MyMethod();
proxy.Close();
```

输出

```
MyService.MyService() Counter = 1 Counter = 2 MyService.Dispose()
```

3.单例服务

在此活化的WCF模式下，所有客户端请求独立于彼此，它们到服务端点的连接会连接到相同的单实例。只有当主机关机那么单例服务得不到处理。

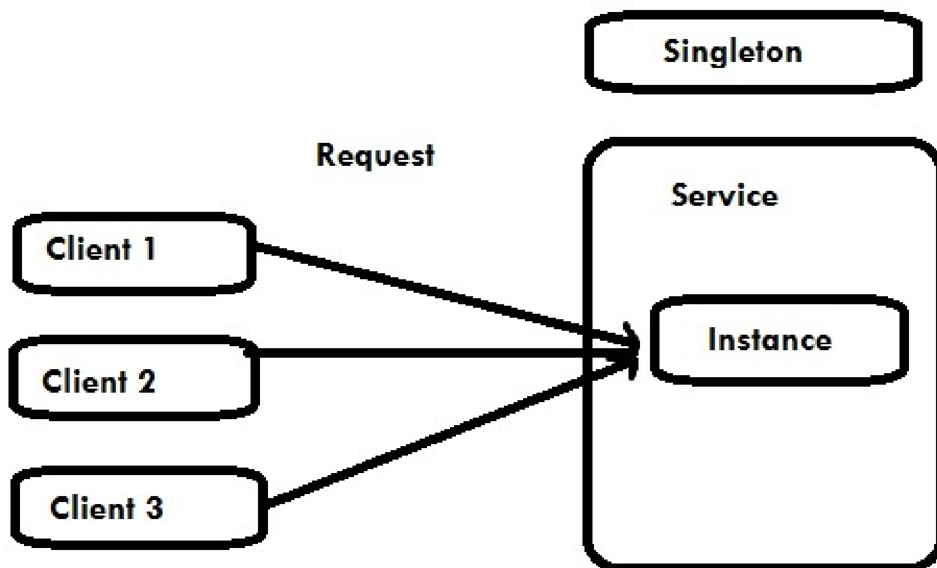
这项服务是刚刚创建创建主机时。在这种情况下，主机不提供任何单一实例，该服务将返回为NULL。激活模式是最好的时候，每个方法调用的工作量少，无等待的操作后台有没有在运行。

InstanceContextMode属性需要设置为InstanceContextMode.Single启动这一单例服务。

因此，一个单例服务可以被配置为

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
class MySingleton : ...
{...}
```

单例服务的过程可以被描述如下图所示。



单例代码实例的初始化和托管

服务代码

```
[ServiceContract]
interface IMyContract
{
  [OperationContract]
  void MyMethod( );
}
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]
class MySingleton : IMyContract
{
  int m_Counter = 0;
  public int Counter
  {
     get
     {
        return m_Counter;
     }
     set
     {
        m_Counter = value;
     }
  }
  public void MyMethod( )
  {
     m_Counter++;
     Trace.WriteLine("Counter = " + Counter);
  }
}
```

主机代码

```
MySingleton singleton = new MySingleton( );
singleton.Counter = 42;

ServiceHost host = new ServiceHost(singleton);
host.Open( );
//Do some blocking calls then
host.Close( );
```

客户端代码

```
MyContractClient proxy = new MyContractClient( );
proxy.MyMethod( );
proxy.Close( );
```

输出

```
Counter = 43
```

WCF事务 - WCF教程

事务处理在WCF(Windows Communication Foundation)是一套遵循一些性质，统称为ACID的操作。这里，如果一个操作出现故障，整个系统就会自动失败。如网上订单生成，就可能使用事务。下面的例子可以帮助理解事务的过程中更简单的术语。

例子

假设一台液晶电视是您从在线商店订购，你会通过信用卡支付的金额。当输入必要的信息来下订单，同时出现两个操作。一个特定的量被从您的银行账户中扣除，第二是供应商贷记相同。两个操作必须以有一个成功的事务成功执行。

WCF事务属性

WCF事务有以下的四个属性。

- 原子性 – 所有的操作都必须作为在完成一个事务的一个不可分割的操作。
- 一致性 – 无论是什么操作设置，系统始终处于的状态总是按照预期的，即事务的结果一致性。
- 隔离性 – 系统的中间状态是不可见的外部世界的任何实体，直到交易完成。
- 持久性 – 提交状态保持无论任何形式的故障（硬件，停电等）。

在配置一个WCF事务，有一些因素需要考虑。这些约束力和操作行为。

- 绑定 – 支持事务的WCF绑定只有几个，这是至关重要的，从只有这些绑定，默认情况下处于禁用状态，并应能获得事务所需支持做出选择。这些绑定说明如下。
    - NetTcpBinding
    - NetNamedPipeBinding
    - WSHttpBinding
    - WSDualHttpBinding
    - WSFederationHttpBinding
- 操作行为 – 同时结合促进事务传播的路径，操作负责和操作配置的是至关重要的。两个属性的主要用途是相同的。它们是TransactionFlow和TransactionScopeRequired。这里应当注意的是，TransactionFlow属性主要具有三个值，它们是Allowed, Mandatory 和 NotAllowed。

下面的代码显示了改变的约束力和事务约定配置是否有利于客户的传播规范。

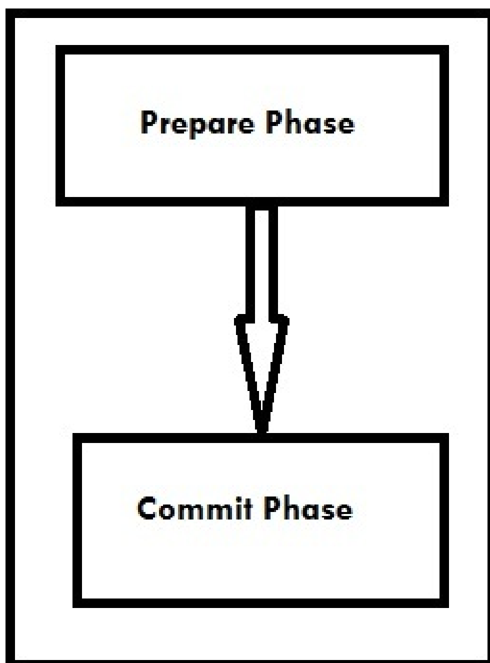
```
<bindings>
  <wsHttpBinding>
     <binding name ="MandatoryTransBinding" transactionFlow ="true">
        <reliableSession enabled ="true"/>
     </binding>
  </wsHttpBinding>
</bindings>
```

事务协议

WCF使用三种协议事务，这些都是轻量级，旧事务和WS-原子事务（WS-AT）。WS-AT是一种可互操作协议，可以跨防火墙的流量分布式事务。然而，该协议不应当事务是严格基于微软技术使用。

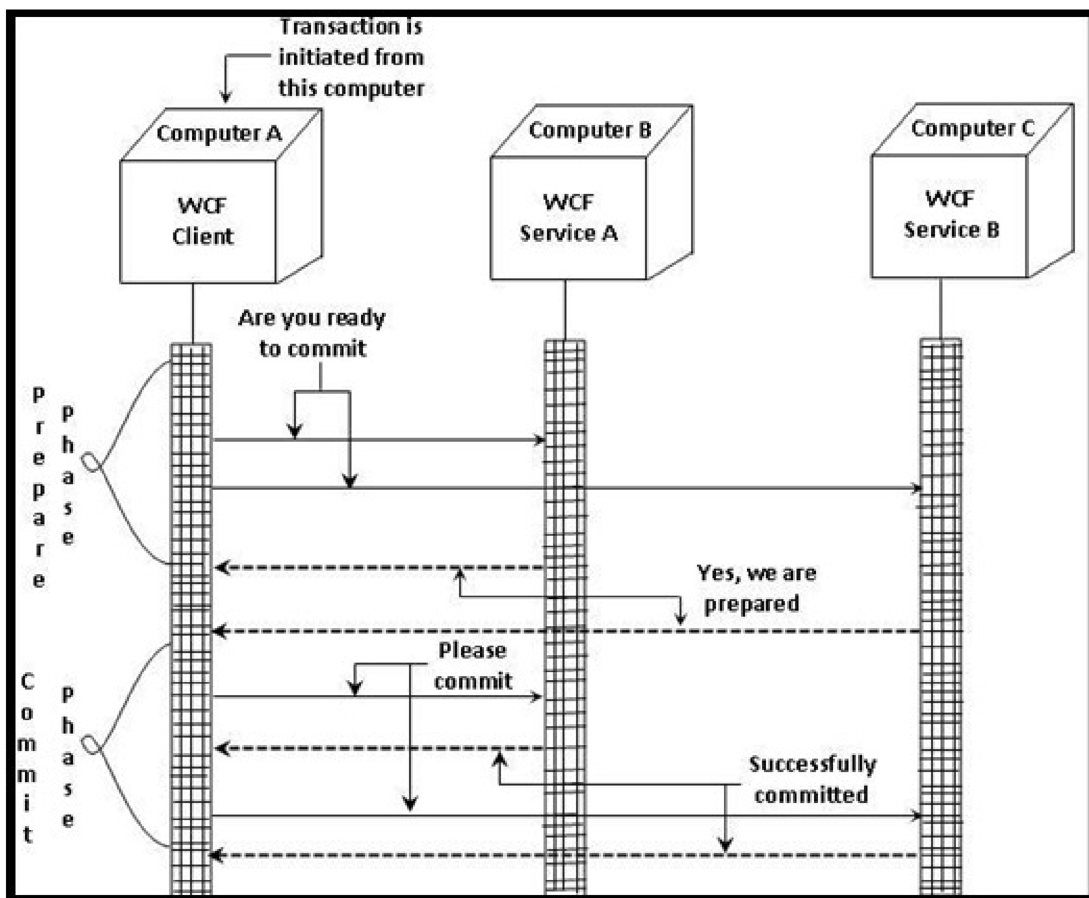
WCF事务阶段

有两个阶段一个WCF事务，如下所述。



- 准备阶段 - 在这个阶段，事务管理器检查是否所有的实体准备好提交的事务。
- 提交阶段 - 在这个阶段，实体的提交在现实中得到开始。

理解一个WCF事务的两相的功能，让我们看看下面的图。



启用WCF事务处理

要成功地使一个WCF事务成功，需要遵循一系列的六个步，。必要的步骤如下所示。

步骤**1**：创建两个**WCF**服务

在这方面，最重要的一步是建立在WCF中两个服务项目，参与到一个事务。数据库事务将在这两个服务的执行，并且应当理解，它们是如何被一个WCF事务统一。 WCFTransactions的web应用程序也被创建在单个事务范围占用两个创建的服务。



第**2**步：创建方法并且其属性有**TransactionFlow**属性

这里，UpdateData方法将被创建为将WCF服务插入到具有OperationContract特性的数据库。为了完成这个任务，接口类ServiceContract首先创建。用于实现该事务在新创建的方法，它具有TransactionFlow属性和事务都使用相同的值。

```
[ServiceContract]
  public interface IService1
  {
     [OperationContract]
     [TransactionFlow(TransactionFlowOption.Allowed)]
     void UpdateData();
  }
```

第3步：WCF服务带有TransactionScopeRequired属性的实现

由下面所示的编码完成。

```
[OperationBehavior(TransactionScopeRequired = true)]
public void UpdateData()
{
  try
  {
     SqlConnection objConnection = new SqlConnection(strConnection);
     objConnection.Open();
     using(SqlTransaction transaction = Program.dbConnection.BeginTransaction())
     {
         Boolean doRollback = false;
         using(SqlCommand cmd = new SqlCommand("insert into Customer (Customer name, Cust
         try
         {
           cmd.ExecuteNonQuery();
         }
         catch(SqlException)
         {
           doRollback = true;
           break;
         }
     }
     if(doRollback)
        transaction.Rollback();
     else
        transaction.Commit();

  }
  finally
  {
      objConection.Close();
  }
}
```

步骤4：由WCF服务配置文件启用事务流程

相同编码如下面给出：

```
<bindings>
  <wsHttpBinding>
     <binding name="TransactionalBind" transactionFlow="true"/>
  </wsHttpBinding>
</bindings>
```

重要的是要重视事务允许的终点绑定暴露WCF服务。

```
<endpoint address="" binding="wsHttpBinding"
    bindingConfiguration="TransactionalBind" contract="WcfService1.IService1">
```

第**5**步：在一个事务中调用两个服务

这里，上述两种服务被称为在一个事务中，为此目的，所述的TransactionScope对象用于这两个服务。上述对象的完整方法被调用来提交WCF事务。如果回滚，那么Dispose方法被调用。

```
using (TransactionScope ts = new TransactionScope(TransactionScopeOption.RequiresNew))
{
  try
  {
     // Call your webservice transactions here
     ts.Complete();
  }
  catch (Exception ex)
  {
     ts.Dispose();
  }
}
```

以下的小片的完整代码，其中WCF提交数据已经被分组描述在一个范围。

```
using (TransactionScope ts = new TransactionScope(TransactionScopeOption.RequiresNew))
{
  try
  {
     ServiceReference1.Service1Client obj = new ServiceReference1.Service1Client();
     obj.UpdateData();
     ServiceReference2.Service1Client obj1 = new ServiceReference2.Service1Client();
     obj1.UpdateData();
     ts.Complete();
  }
  catch (Exception ex)
  {
     ts.Dispose();
  }
}
```

第6步：测试WCF事务

测试是在第6步，也是最后一步并调用第1个WCF服务后，发生异常（被迫）。

```
using (TransactionScope ts = new TransactionScope(TransactionScopeOption.RequiresNew))
{
    try
    {
        ServiceReference1.Service1Client obj = new ServiceReference1.Service1Client();
        obj.UpdateData();
        throw new Exception("There was a error here , revert the first service entry");
        ServiceReference2.Service1Client obj1 = new ServiceReference2.Service1Client();
        obj1.UpdateData();
        ts.Complete();
    }
    catch (Exception ex)
    {
        ts.Dispose();
    }
}
```

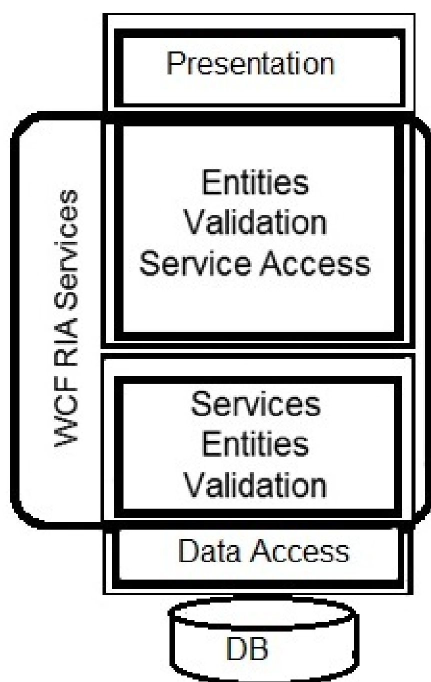
This database entry will be rolled back

Forced a exception

WCF RIA服务 - WCF教程

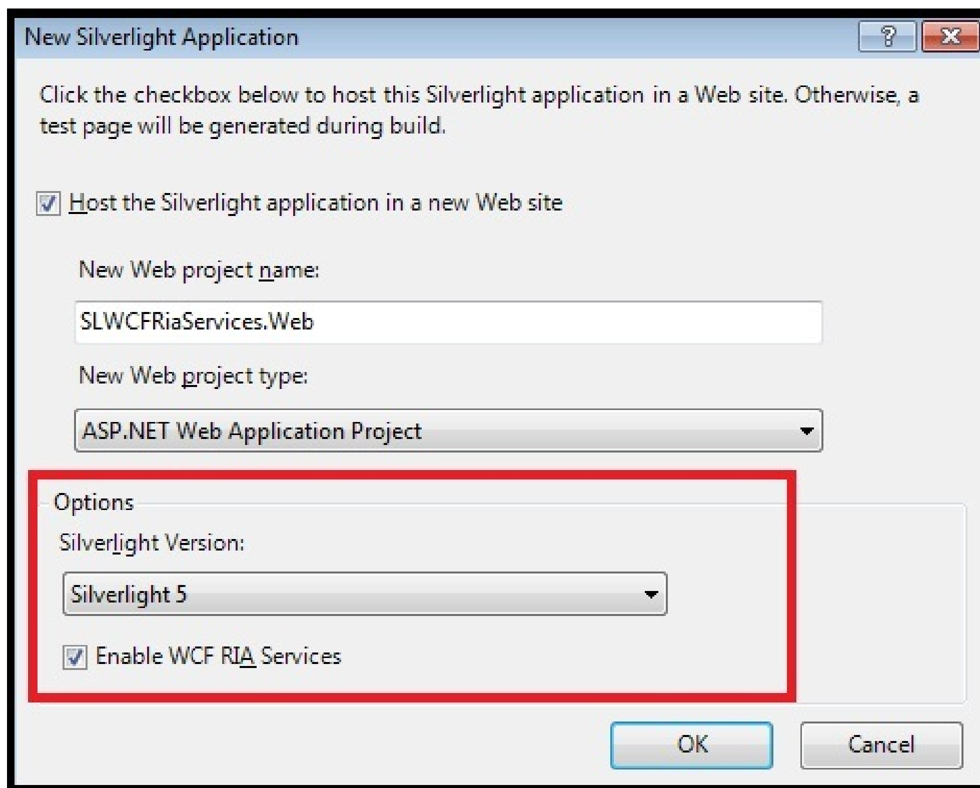
WCF RIA Service是更高层次的框架，像.NET 4和Silverlight4框架，简化构建在Silverlight中一个复杂的业务应用程序通过提供客户端验证的过程的新的组成部分。 RIA代表富Internet应用程序。这里必须注意的是，提供的微软，Silverlight是一个框架，理想的富互联网应用程序，并且可以作为浏览器插件，和Adobe Flash一样使用。

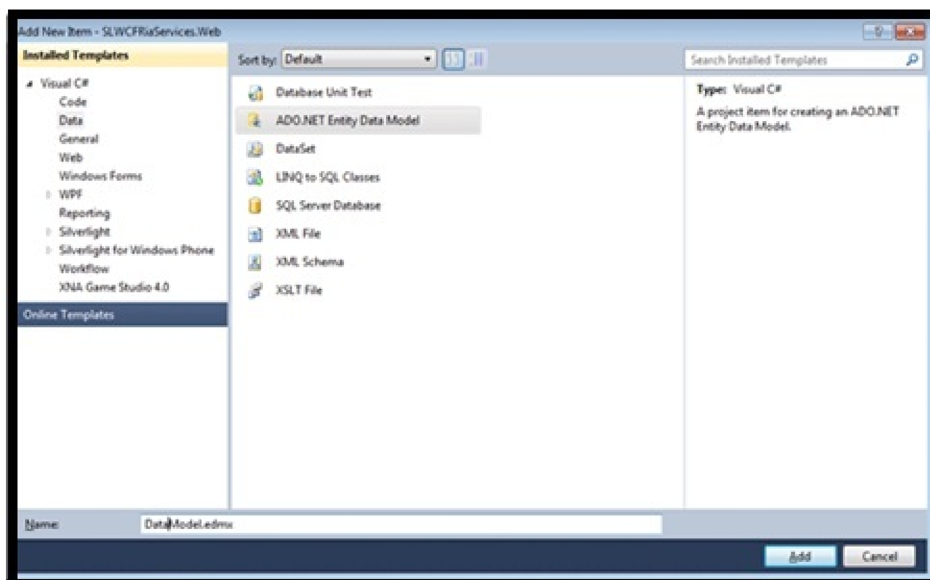
WCF RIA服务主要是基于WCF服务的标准版本。要了解有关WCF RIA Services的更好的方式，如下图所示的架构，WCF RIA服务有重点。 DB在这里为数据库中的缩略形式。



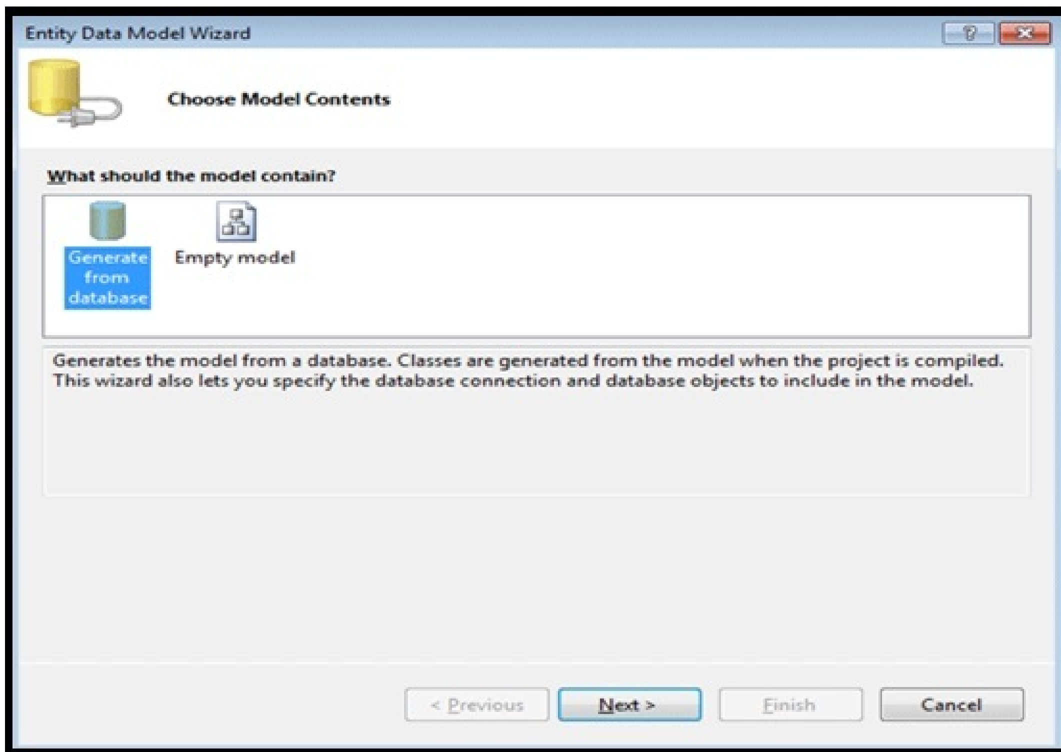
创建WCF RIA Service在下一步会有一个更深入的了解。按照下面给出的按部就班地进行就可以了。

步骤1：使用Silverlight5创建名为SLWCFRiaServices.Web的一个新的Web项目，然后选择ADO.NET实体数据模型，以相同的添加一个新的项目。

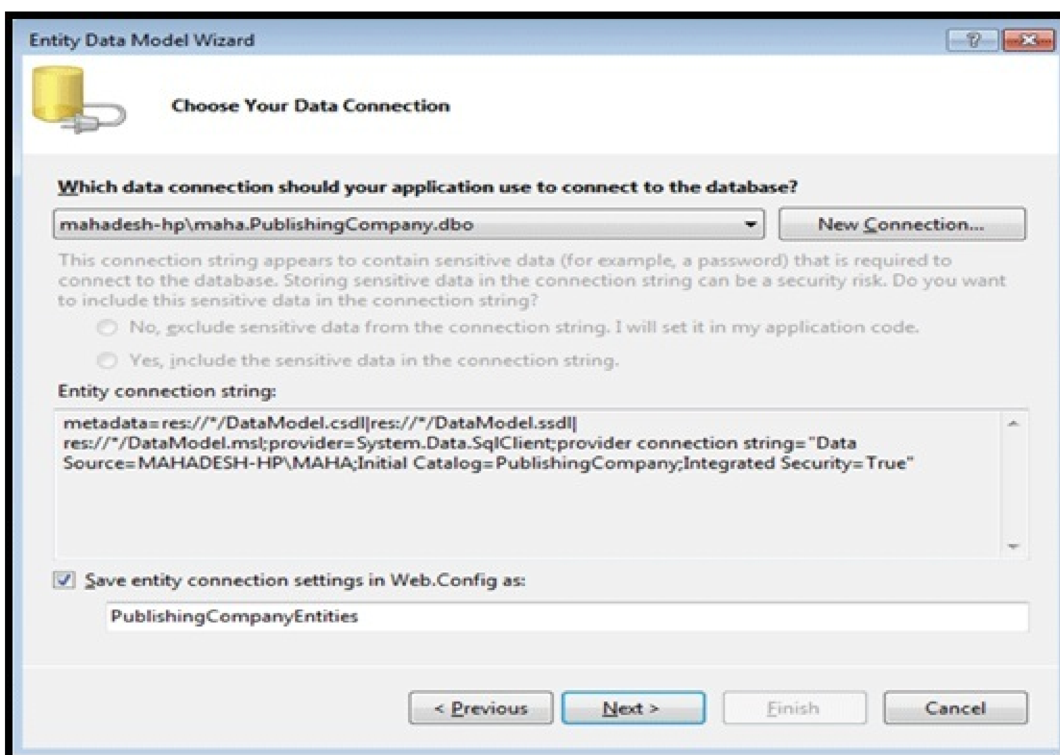


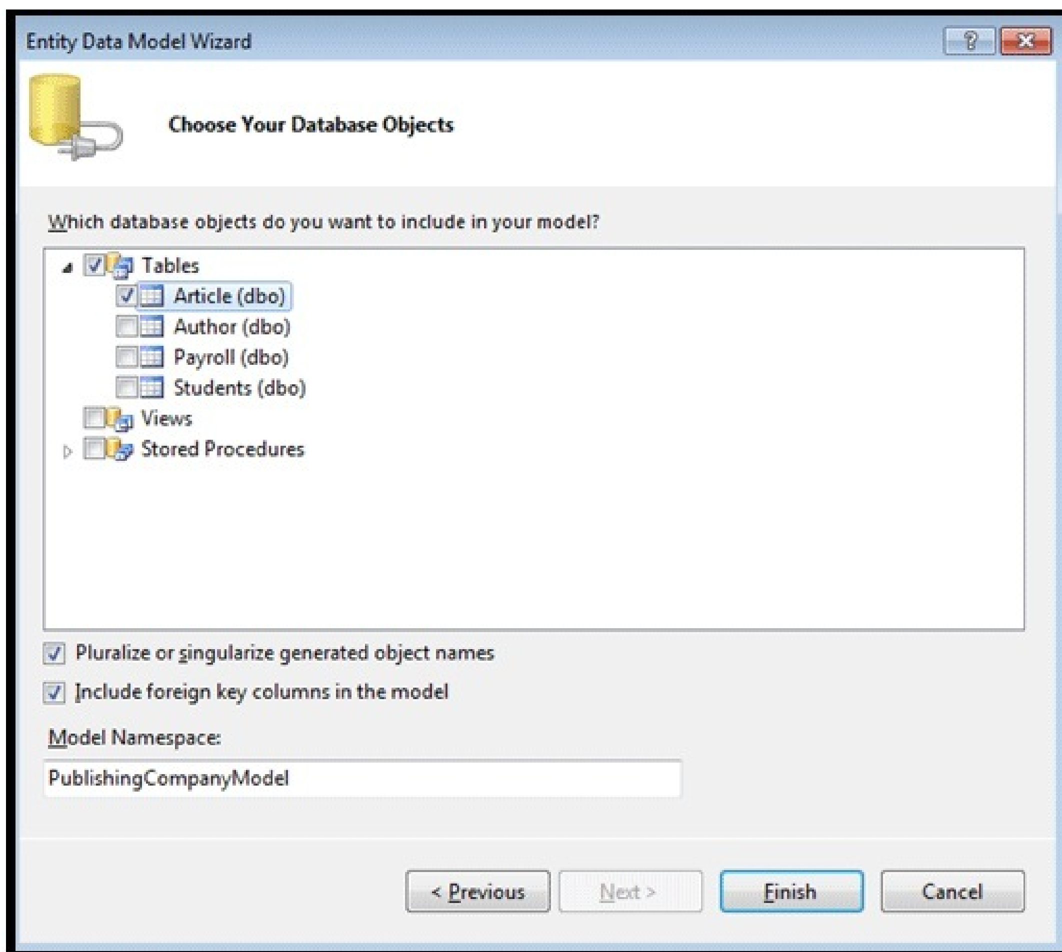


第2步：现在，通过生成从数据库模型选择的实体数据模型向导模式的内容。

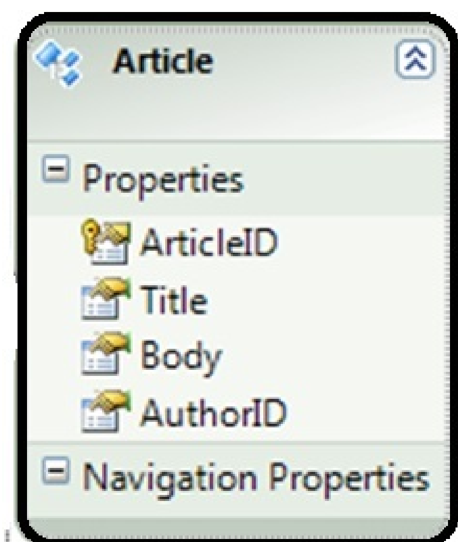


步骤3：从同一个向导，请选择数据连接和数据库对象。

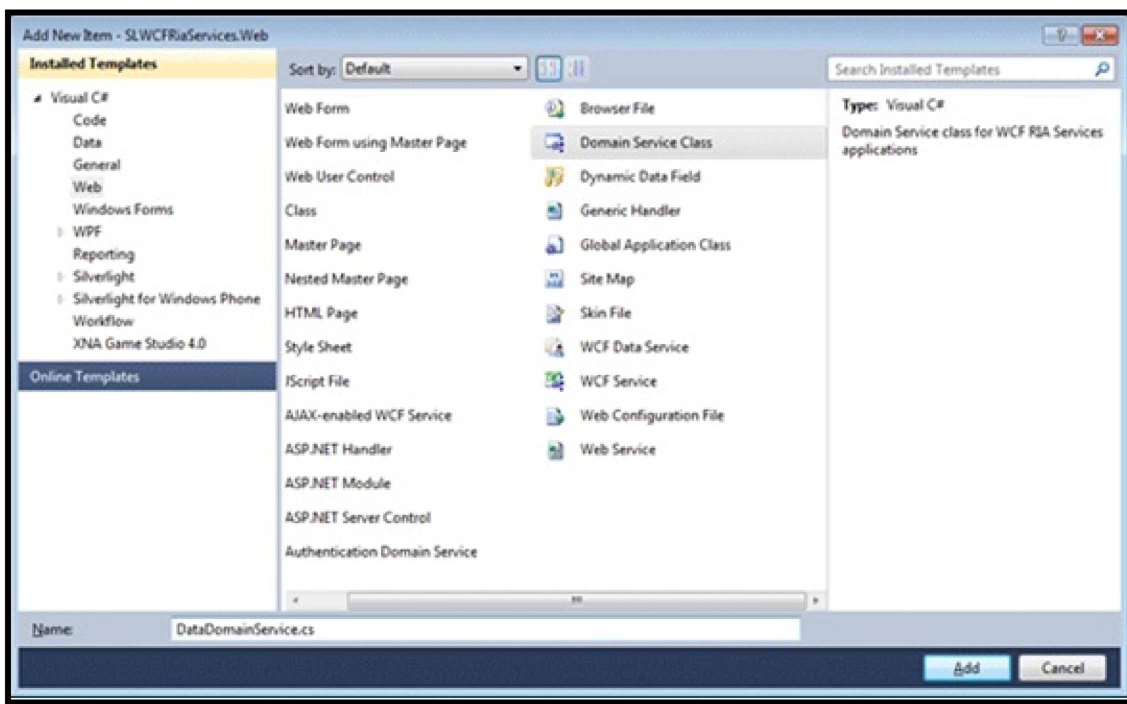


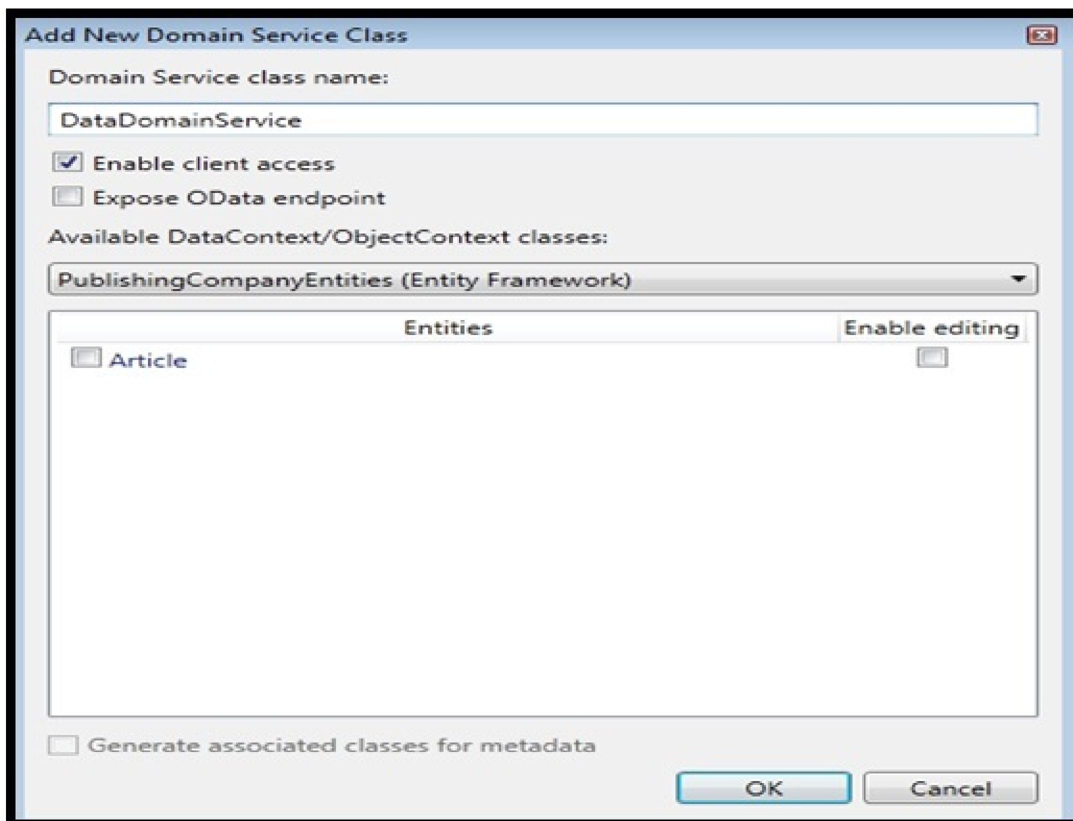


第4步：生成解决方案，以便在未来的认识的数据模型是不是要创建的域名服务问题。

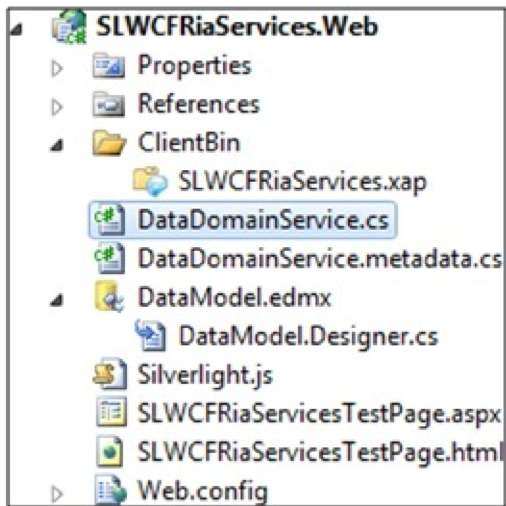


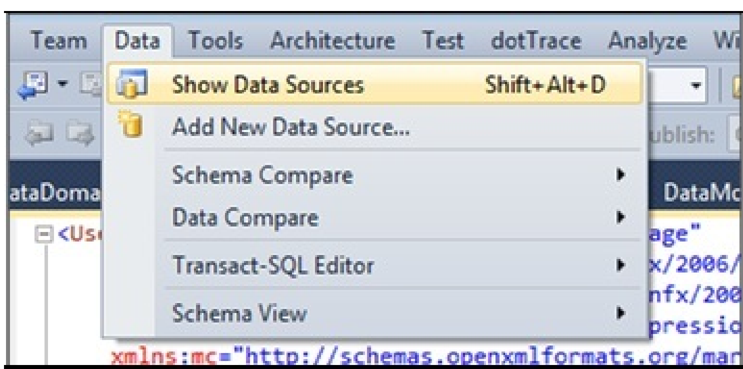
第5步：现在，通过添加新的项目创建在Web项目中的域名服务，确保让客户端访问。



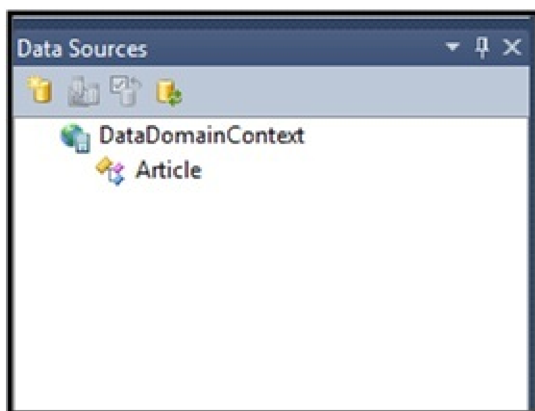


步骤6：在紧接着的下一个步骤，产生了一些类的会发生，因此有必要再次构建它们。

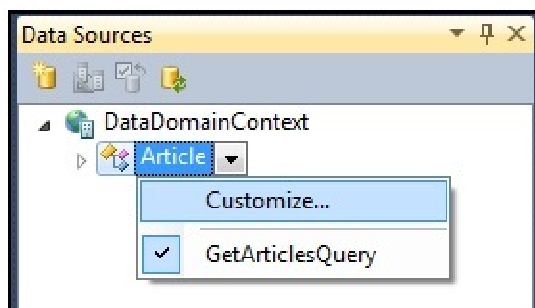




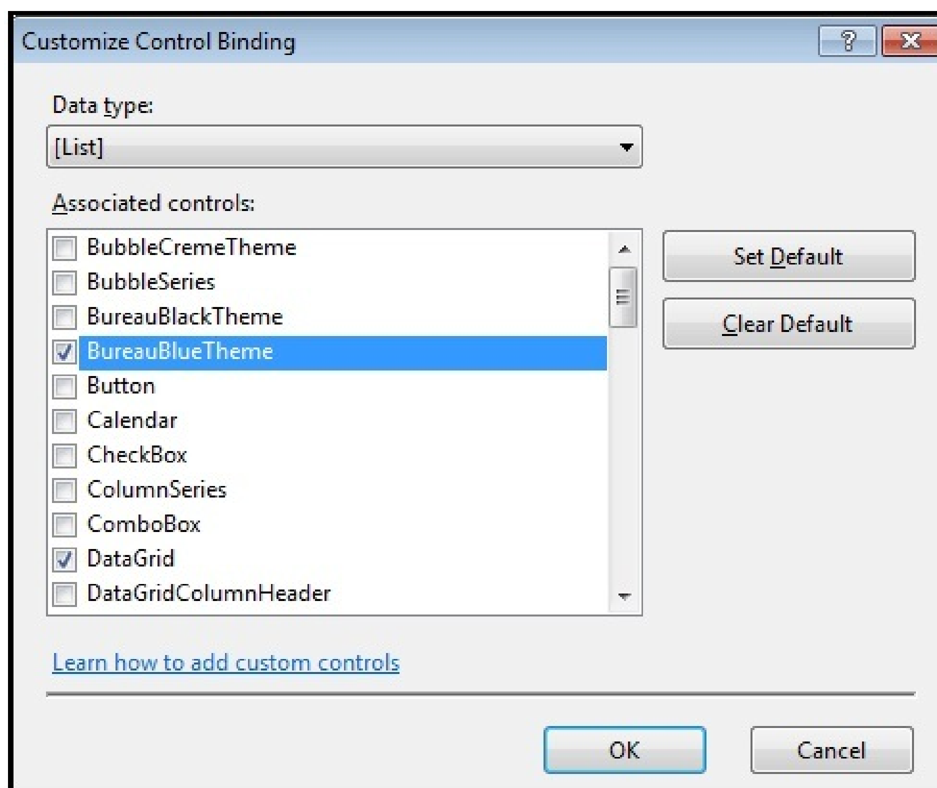
第7步：在这一步，DataDomainContext示出了数据源面板。



第8步：下面DataDomainContext文章应选择并应进行定制。

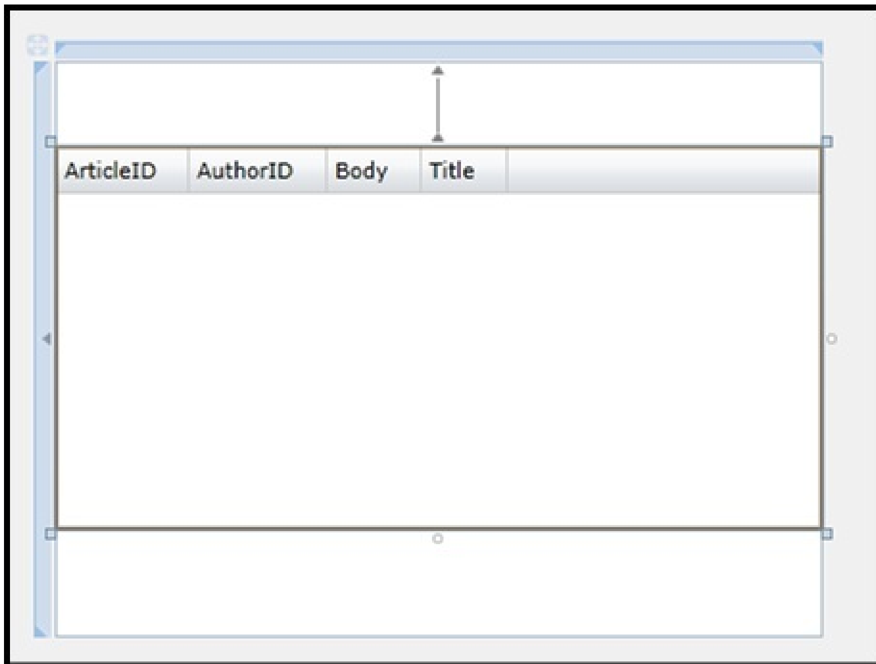


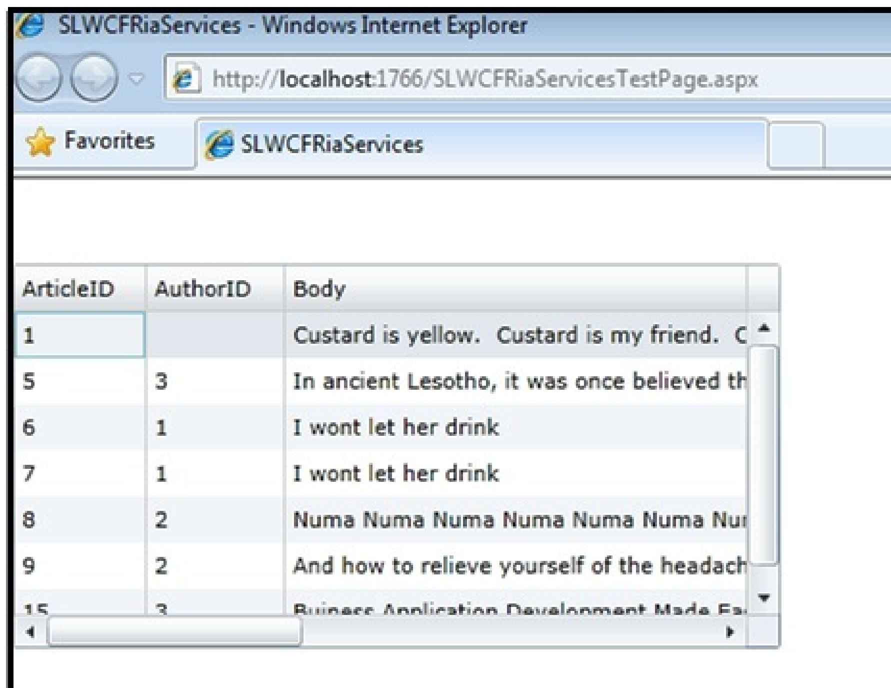
第9步：连接DataGrid控件的数据源是在这里承诺以及选择的主题，如在此步骤BureauBlue主题已被选中。





步骤10：最后一个和最后步骤包括将要设计的屏幕，并通过简单的拖放添加实体在MainPage布局面积。同样重要的是要确保AutoGenerateColumns="true"，并运行它来查看输出。





先决条件

有一些先决条件经历WCF RIA服务的攻略，如下面。

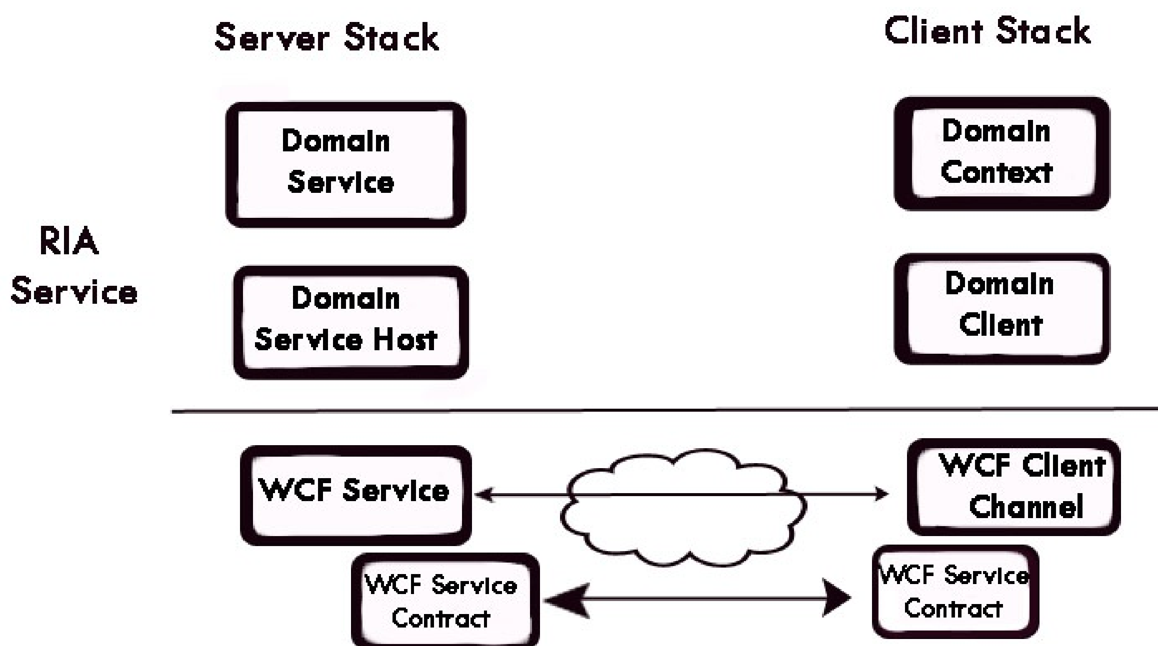
- Visual Studio 2010/ Visual Studio 2012
- Silverlight Developer Runtime
- Latest version of RIA Services Toolkit
- SDK (Software Development Kit)

WCF RIA域名服务

一个域的服务包括一组相关的业务数据操作，并没有什么，但它暴露任何WCF RIA服务应用程序的业务逻辑WCF服务。

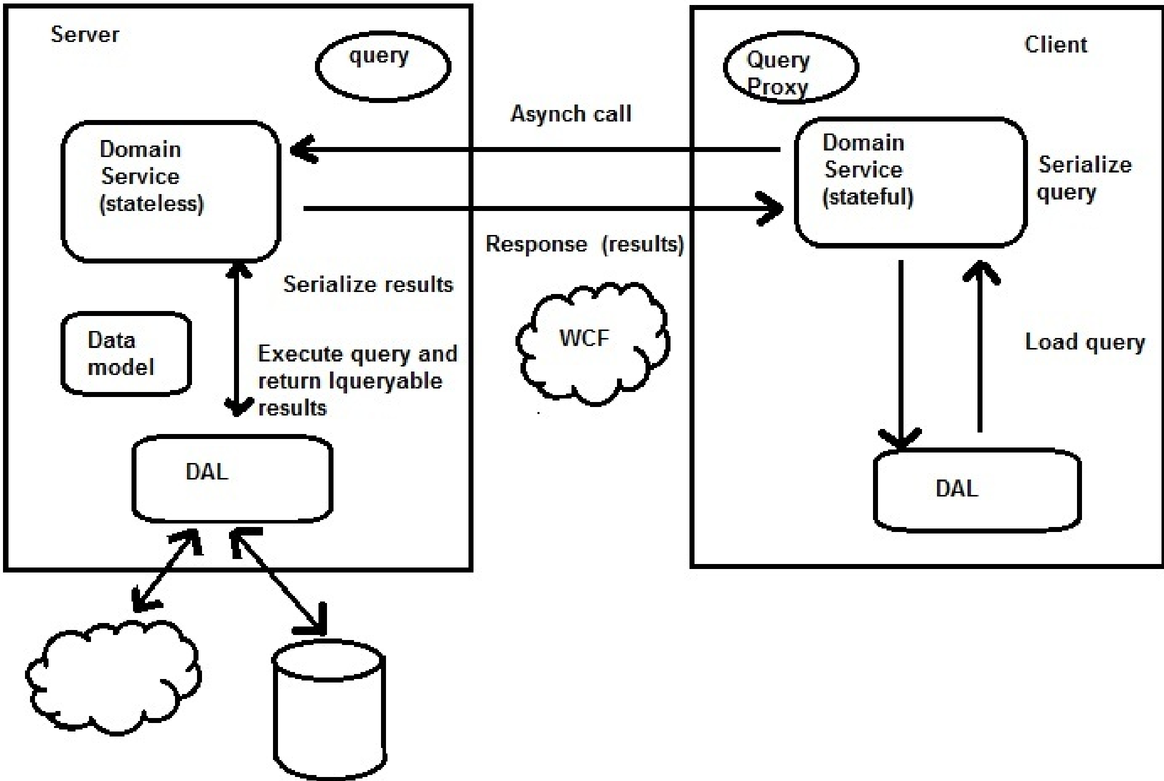
一个WCF RIA域名服务有内部托管类DomainServiceHost又使用WCF的ServiceHost类的托管应用程序。为了让域名访问服务的客户端项目，它应该有EnableClientAccessAttribute属性。每当一个新的域服务类添加属性得到自动应用。

下图显示了WCF RIA域名服务的体系结构



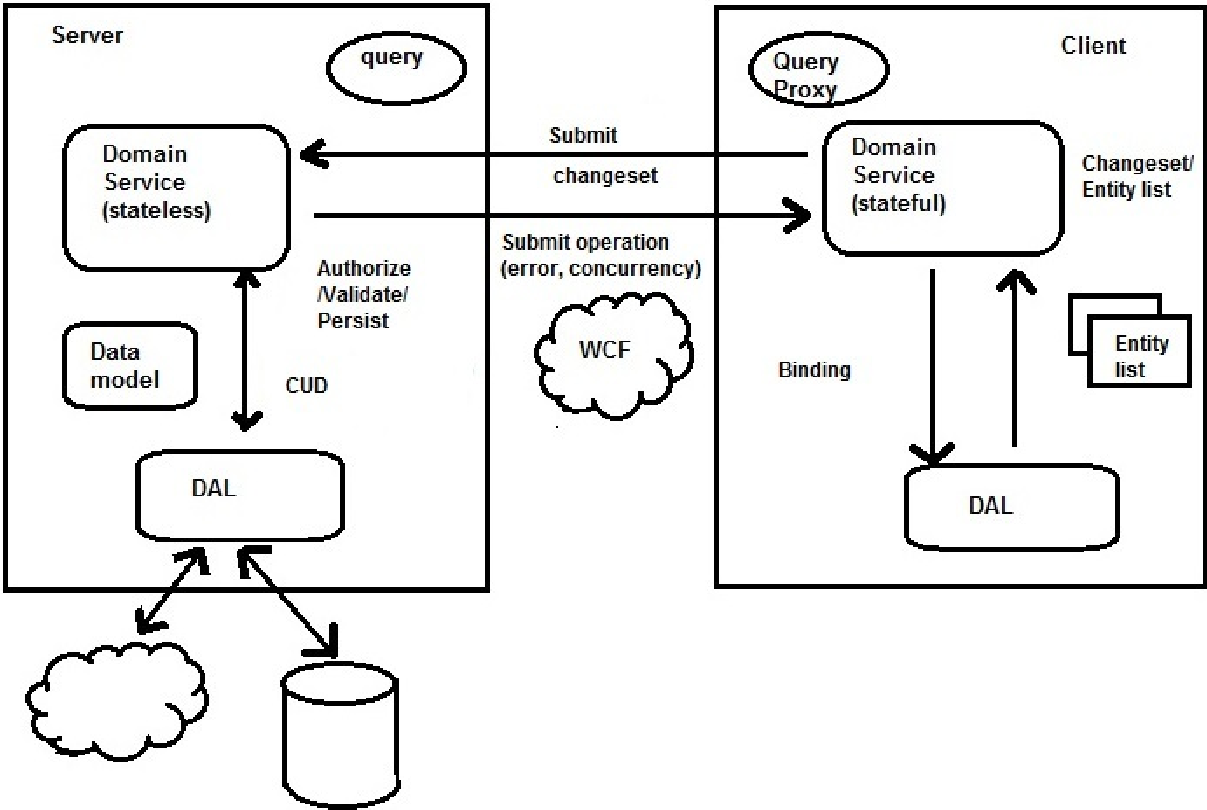
WCF RIA Services数据查询

下面的图显示了如何将查询客户端上创建和在服务器侧执行返回Jqueryable结果。但必须注意的是，DAL是这里的数据访问层。



WCF RIA Services更新数据

该图显示了数据是如何通过执行CUD更新服务器端（创建，更新，删除）操作。这里应注意的是，WCF RIA服务总是无状态的服务器端。

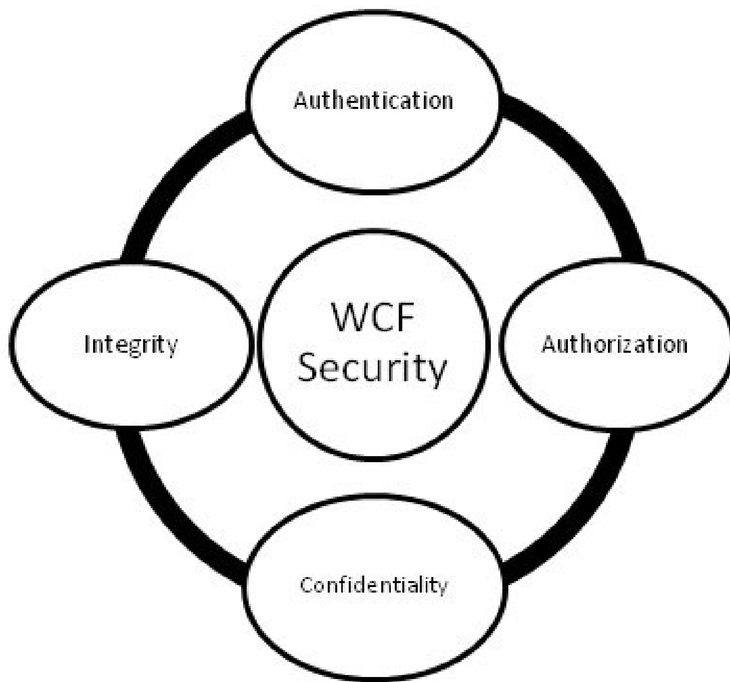


WCF安全 - WCF教程

一个强大的WCF服务安全系统，拥有两种安全模式或级别预期的客户端可以访问的服务。这是常见的分布式事务的安全威胁正在放缓，在很大程度上由WCF决定。

关键的安全功能

WCF服务有四个主要的安全功能，如下图所示。



- 认证- 这里认证是不限定于识别消息的发送者，但是相互的，即消息接收器的认证是必需的，以排除任何种类的中间人攻击的可能性。
- 授权- 这是采取了一个WCF服务，以确保安全性的下一步骤，并在此确定服务是否应授权调用方进一步或不会进行。虽然授权不依赖于身份验证时，它通常如下认证。
- 保密 - 调用者和服务之间的信息交流是保密的，限制其解释并不打算为其他人公开消息。为了使这成为可能，加密与各种各样的其他机制一起使用。
- 完整性- 最后一个关键概念是保持完整性，即提供了保证，该消息尚未从发送者到接收者不被任何人在这一过程篡改。

传输安全模式

WCF提供以下传输安全模式，以确保在客户机和服务器之间的安全通信。多样传输安全模式如下所述。

- **None** - 此模式不能保证任何消息安全和服务不获取有关客户端的任何凭据。这种模式是非常危险的，因为它可以使信息被篡改，因此不推荐使用。

```
<wsHttpBinding>
   <binding name="WCFSecurityExample">
      <security mode="None"/>
   </binding>
</wsHttpBinding>
```

- **Transport** - 这种模式是实现信息通过使用通信协议，如TCP，IPC，HTTPS和MSMQ一个安全的传输的最简单方法。这种模式是比较有效的，当在传输点至点，并主要是用于在受控环境中，也就是说，内部网应用。

```
<wsHttpBinding>
   <binding name="WCFSecurityExample">
      <security mode="Transport"/>
   </binding>
</wsHttpBinding>
```

- **Message** - 安全模式可以相互验证，并提供隐私的消息进行加密，并且可以通过http，这不被认为是一种安全协议被传输在很大程度上。这里的安全性提供了端 - 端，而不考虑有多少中介参与消息传送和是否有一个安全的运输或没有。该模式是通过互联网应用程序通常使用。

```
<wsHttpBinding>
   <binding name="WCFSecurityExample">
      <security mode="Message"/>
   </binding>
</wsHttpBinding>
```

- **Mixed** - 这种安全模式是不经常使用，客户端身份验证仅在客户端级别提供。

```
<wsHttpBinding>
   <binding name="WCFSecurityExample">
      <security mode="TransportWithMessageCredential"/>
   </binding>
</wsHttpBinding>
```

- **Both** - 此安全方式包括两种传输安全性和信息的安全性，提供了健壮的安全盖，但通常会导致超载的整体性能。这一个仅由MSMQ支持。

```
<netMsmqBinding>
   <binding name="WCFSecurityExample">
      <security mode="Both"/>
   </binding>
</netMsmqBinding>
```

所有的WCF绑定，除非有basicHttpBinding传输安全性默认情况下有一定关系。

消息安全级别

消息级安全性不依赖于WCF协议。它是通过使用一个标准的算法对数据进行加密采用与消息数据本身。有若干客户端凭证可用于不同的绑定的消息的安全级别，这些将在下面讨论。

WCF消息级安全性的客户端证书

None :在此，使用加密来保护该消息而被执行，这意味着，该服务可以由一个匿名客户访问没有客户机认证。除了basicHttpBinding，所有的WCF绑定支持此客户端凭据。然而，应当注意的是，对于NetNamedPipeBinding客户端凭证不可用。

- Windows - 在这里无论是信息的加密和认证的客户端发生了一个实时登录的用户。在此情况下，也不同于所有其他的WCF绑定，NetNamedPipeBinding不可用以及basicHttpBinding不提供支持。
- UserName - 这里消息被加密，以及通过提供用户名固定，而客户端进行认证，因为它们需要提供密码。 basicHttpBinding就像上面的两个客户端凭证，不支持用户名和它不适用于NetNamedPipeBinding。
- Certificate - 随着信息加密，客户端和服务获得与证书的身份验证。此客户端证书可用，并且支持所有的WCF绑定，除了NetNamedPipeBinding。
- IssuedToken - 类似CardSpace从一个机构颁发的令牌用于验证的消息。这里也进行消息的加密。

下面的代码显示了客户端凭据如何配置WCF的信息安全等级/模式。

```
<netTcpBinding>
   <binding name="WCFMessageSecurityExample">
      <security mode="Message">
         <message clientCredentialType="None"/>
      </security>
   </binding>
</netTcpBinding>

<netMsmqBinding>...</netMsmqBinding>
</bindings>
<behaviors>...</behaviors>
```

这里，必须指出的是，传输安全模式具有超过该消息的安全级别的边缘，因为前者是更快。它不需要任何额外的编码，并提供互操作性的支持，并且因此不会降低整体性能。

然而，从安全角度考虑，将消息安全模式是更加健壮，并且独立的协议，并提供端到端的安全性。

WCF异常处理 - WCF教程

WCF服务开发者可能会遇到需要以适当的方式向客户端报告一些不可预见的错误。这样的错误，称为异常，通常是通过使用try/catch块来处理，但同样，这是非常具体的技术。

由于客户端的关注领域不是关于如何发生错误或因素导致的错误，SOAP错误的约定，用于从WCF服务的传送到客户端的错误消息。

故障分析合约使客户端能够发生在一个服务错误的文件视图。下面的例子给出了一个更好的了解。

步骤1：一个简单的计算器服务与除法运算，将创建一般常见的异常。

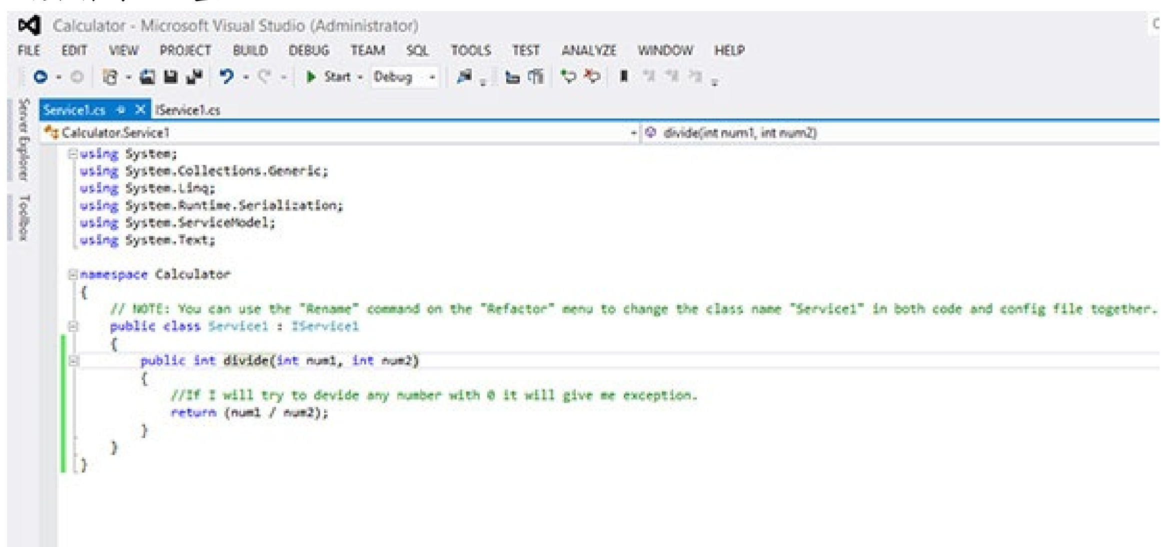
```
using System;
usingSystem.Collections.Generic;
usingSystem.Linq;
usingSystem.Runtime.Serialization;
usingSystem.ServiceModel;
usingSystem.Text;

namespace Calculator
{
   // NOTE: You can use the "Rename" command on the "Refactor" menu to change
   // the interface name "IService1" in both code and config file together.

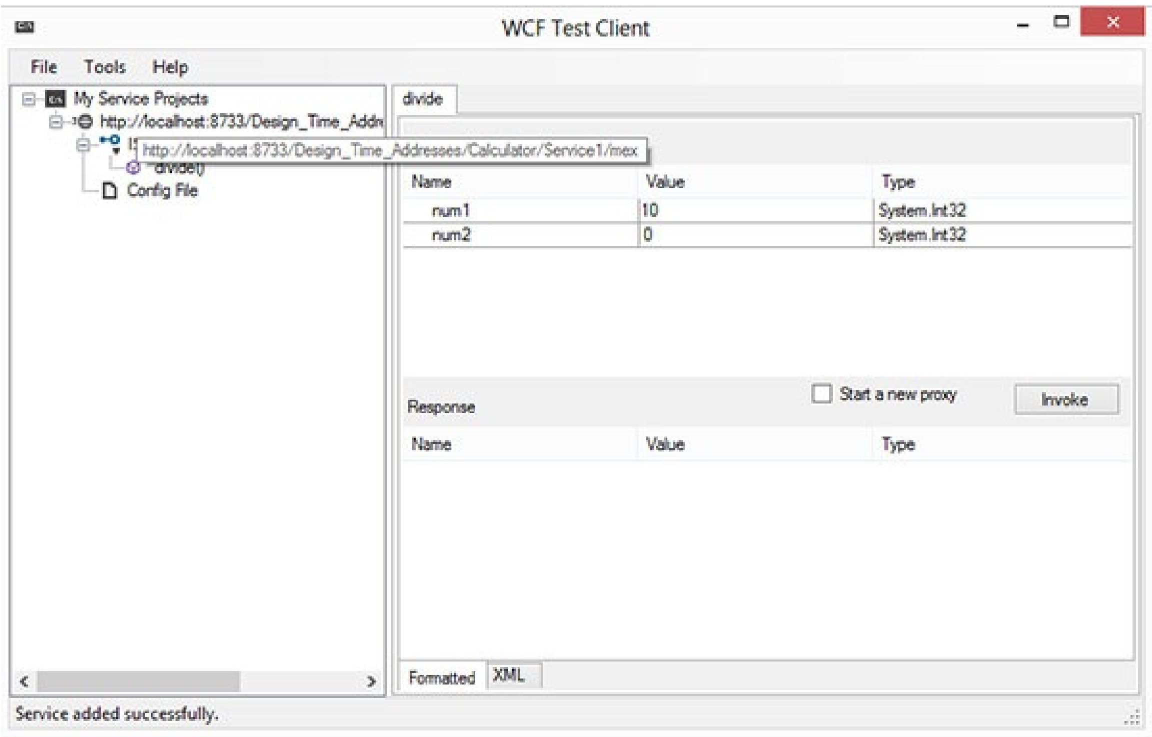
   [ServiceContract]

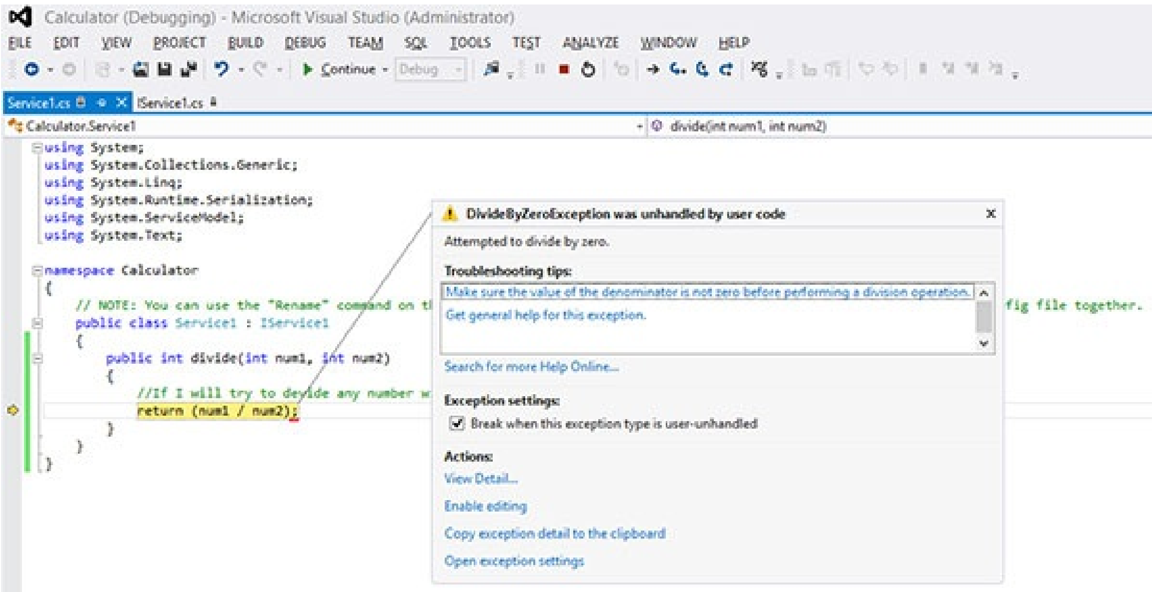
   public interface IService1
   {
      [OperationContract]
      int divide(int num1, int num2);
      // TODO: Add your service operations here by www.yiibai.com
   }
}
```

该类编码文件显示如下：

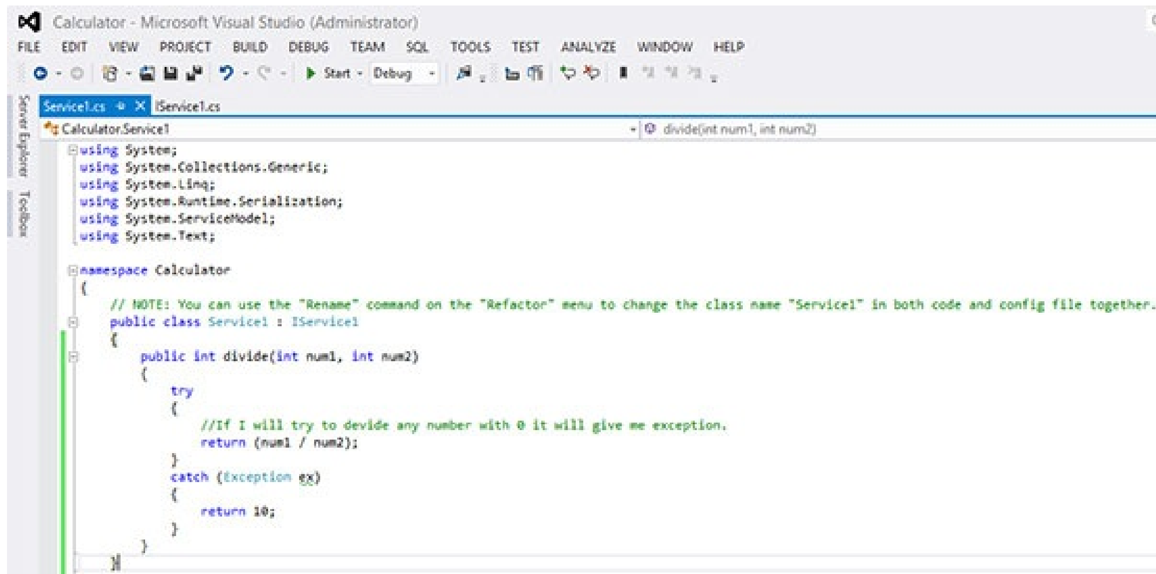


现在，当我们试图让10除以零，计算服务将抛出一个异常。

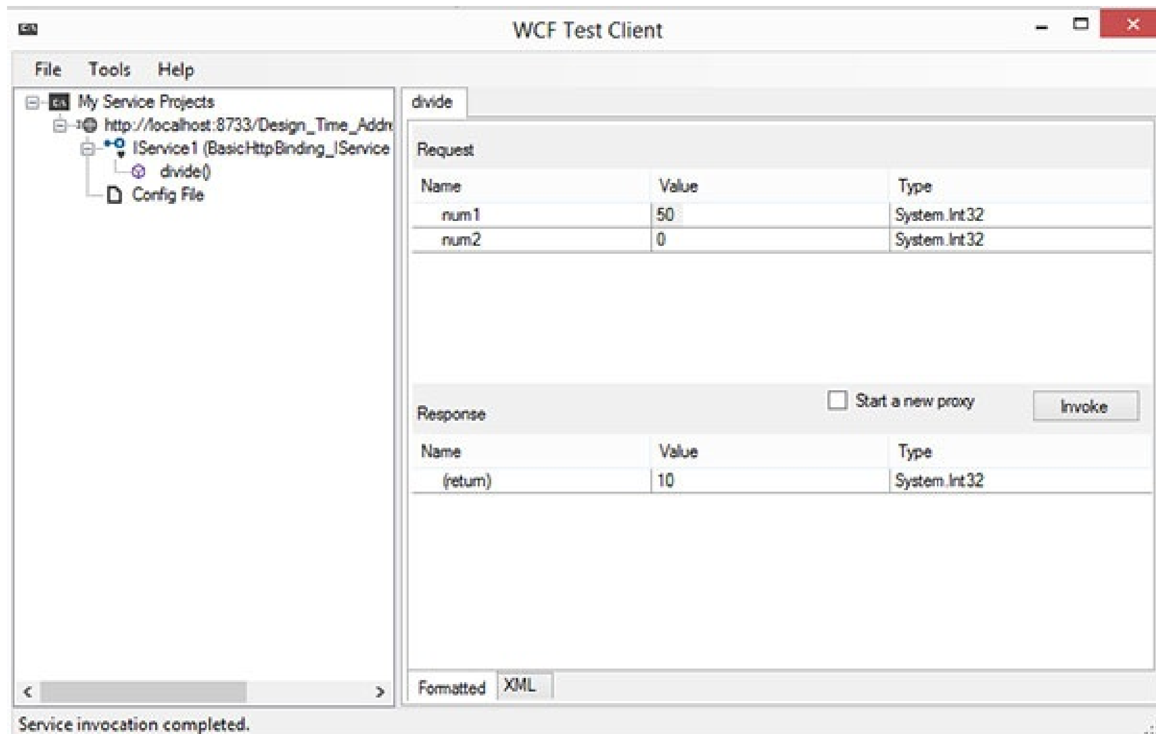




该异常可以通过try/catch块来处理。



现在，当我们试图让任何整数除以0，它会因为我们在catch块中处理其返回值10。



步骤-2：FaultException异常用于在该步骤中进行通信的异常信息从服务客户端返回。

```
public int Divide(int num1, int num2)
{
   //Do something
   throw new FaultException("Error while dividing number");
}
```

file:///F:/WCF/MyCalculatorWCFServiceHost/MyCalculatorWCFServiceHost/bin/...

```
Service is host at 05-10-2014 8.40.00
Error while dividing number.
```